BAOU Education for All

DR.BABASAHEB AMBEDKAR
OPEN UNIVERSITY

BCA

BACHELOR OF COMPUTER APPLICATION

BCAR-204

Object Oriented Concept & Programming-I (Core Java)

OBJECT ORIENTED CONCEPTS & PROGRAMMING–1 (CORE JAVA)

BAOU
Education
for All

DR. BABASAHEB AMBEDKAR OPEN UNIVERSITY
AHMEDABAD

Editorial Panel

Authors : Dr. Ankit R. Bhavsar
Sr. Assistant Professor
Faculty of Computer Application & Information Technology (FCAIT)
GLS University, GLS Campus, Opp. Law Garden, Ellisbridge, Ahmedabad

Dr. Ashish Parejiya
Former Senior Assistant Professor,
Indus University, Ahmedabad.
Project Manager, Piramal Group, Ahmedabad.

Editor : Dr. Kruti Devang Jani
Sr. Assistant Professor
Shri Chimanbhai Patel Post Graduate Institute of Computer Application (CPIMCA), Ahmedabad.

Language Editor : Dr. Jagdish Vinayakrao Anerao
Associate Professor,
Smt A. P. Patel Arts And,
N. P. Patel Commerce College,
Ahmedabad.

ISBN 978-93-91071-00-4

Edition : 2020

Acknowledgment

Every attempt has been made to trace the copyright holders of material reproduced in this book. Should an infringement have occurred, we apologize for the same and will be pleased to make necessary correction/amendment in future edition of this book.

The content is developed by taking reference of online and print publications that are mentioned in Bibliography. The content developed represents the breadth of research excellence in this multidisciplinary academic field. Some of the information, illustrations and examples are taken "as is" and as available in the references mentioned in Bibliography for academic purpose and better understanding by learner.

ROLE OF SELF–INSTRUCTIONAL MATERIAL IN DISTANCE LEARNING

The need to plan effective instruction is imperative for a successful distance teaching repertoire. This is due to the fact that the instructional designer, the tutor, the author (s) and the student are often separated by distance and may never meet in person. This is an increasingly common scenario in distance education instruction. As much as possible, teaching by distance should stimulate the student's intellectual involvement and contain all the necessary learning instructional activities that are capable of guiding the student through the course objectives. Therefore, the course / self–instructional material are completely equipped with everything that the syllabus prescribes.

To ensure effective instruction, a number of instructional design ideas are used and these help students to acquire knowledge, intellectual skills, motor skills and necessary attitudinal changes. In this respect, students' assessment and course evaluation are incorporated in the text.

The nature of instructional activities used in distance education self–instructional materials depends on the domain of learning that they reinforce in the text, that is, the cognitive, psychomotor and affective. These are further interpreted in the acquisition of knowledge, intellectual skills and motor skills. Students may be encouraged to gain, apply and communicate (orally or in writing) the knowledge acquired. Intellectual–skills objectives may be met by designing instructions that make use of students' prior knowledge and experiences in the discourse as the foundation on which newly acquired knowledge is built.

The provision of exercises in the form of assignments, projects and tutorial feedback is necessary. Instructional activities that teach motor skills need to be graphically demonstrated and the correct practices provided during tutorials. Instructional activities for inculcating change in attitude and behavior should create interest and demonstrate need and benefits gained by adopting the required change. Information on the adoption and procedures for practice of new attitudes may then be introduced.

Teaching and learning at a distance eliminates interactive communication cues, such as pauses, intonation and gestures, associated with the face–to–face method of teaching. This is particularly so with the exclusive use of print media. Instructional activities built into the instructional repertoire provide this missing interaction between the student and the teacher. Therefore, the use of instructional activities to affect better distance teaching is not optional, but mandatory.

Our team of successful writers and authors has tried to reduce this.

Divide and to bring this Self Instructional Material as the best teaching and communication tool. Instructional activities are varied in order to assess the different facets of the domains of learning.

Distance education teaching repertoire involves extensive use of self–instructional materials, be they print or otherwise. These materials are designed to achieve certain pre–determined learning outcomes, namely goals and objectives that are contained in an instructional plan. Since the teaching process is affected over a distance, there is need to ensure that students actively participate in their learning by performing specific tasks that help them to understand the relevant concepts. Therefore, a set of exercises is built into the teaching repertoire in order to link what students and tutors do in the framework of the course outline. These could be in the form of students' assignments, a research project or a science practical exercise. Examples of instructional activities in distance education are too numerous to list. Instructional activities, when used in this context, help to motivate students, guide and measure students' performance (continuous assessment)

PREFACE

We have put in lots of hard work to make this book as user-friendly as possible, but we have not sacrificed quality. Experts were involved in preparing the materials. However, concepts are explained in easy language for you. We have included many tables and examples for easy understanding.

We sincerely hope this book will help you in every way you expect.

All the best for your studies from our team!

OBJECT ORIENTED CONCEPTS & PROGRAMMING–1 (CORE JAVA)

Contents

Unit 8 **JAVA COLLECTION FRAMEWORK**

Introduction, Collection Interface, List Interface, LinkedList Class, ArrayList Class, Stack Class, Queue Interface, Set Interface, TreeSet Class, Hashset Class, Map Interface, TreeMap Class, HashMap Class, Iterator

---

BLOCK 3 : **INHERITANCE, EXCEPTION HANDLING AND MULTITHREADING**

Unit 9 **INHERITANCE**

Introduction, Concept of Inheritance, Polymorphism, Final Keyword

Unit 10 **EXCEPTION HANDLING**

Introduction, Types of Exceptions, Uncaught Exception, Using Try and Catch Block, Using Multiple Catch Statements, Using Methods Defined by Exception and Throwable, User Defined Exceptions, Using Throws/ Throw Keyword, Using Finally Keyword, Nested Try Statements

Unit 11 **UTILITIES & MULTITHREADING**

Introduction, Comparing Arrays : Java Util, Creating a Hash Table : Java Util, Multithreading, Thread Life Cycle, The Thread Class and The Runnable Interface, Thread Priorities, Synchronisation, Deadlock, Suspending, Resuming and Stopping Threads

---

BLOCK 4 : **ABSTRACT WINDOW TOOLKIT AND WORKING WITH FILES**

Unit 12 **APPLET**

Introduction, Difference between Applet and Application, Applet Life Cycle, Creating an Applet, Applet Tag, Reading Parameters into Applet, Implementation of Background Colour, Implementation of Font in Applet

Unit 13 **APPLET GRAPHICS**

Introduction, Drawing Line, Drawing Oval, Drawing Circle, Drawing Rectangle, Drawing Arcs, Drawing Polygons, Drawing Polyline, Delegation Event Model

Dr. Babasaheb Ambedkar Open University Ahmedabad

BCAR-204/ DCAR-204

Object Oriented Concepts & Programming–1 (Core Java)

BLOCK 1 : BASIC PROGRAMMING CONCEPTS IN JAVA

BASIC PROGRAMMING CONCEPTS IN JAVA

Block Introduction :

With the advent of internet, Java was widely used. Initially, it was thought to develop a platform–independent language for consumer electronics like washing machines etc. Java became popular because the Web required platform–independent portable programs. In this block we are going for the introduction to Java technology in Unit No. 1 which will make you understand Features of Java, Comparison of Java with C++ and the concept of Garbage Collection and also essentially Creating a java Program with examples.

Unit No. 2 is meant to enhance further level of understanding about Tokens with relevant examples. Data Types in Java, how to declare a variable accordingly Java coding conventions with detail examples. This would bring good amount of understanding amongst the student about Java coding.

Unit No. 3 is explaining the use of Operators. Mainly operators are used to do some mathematical operations on numerical and binary data. It uses with commonly one, two or three operand.

Unit No. 4 is mainly emphasizing and gives further level of understanding about loops, nested loops, selection statements and finally Arrays with their respective examples.

Block Objectives :

After learning this Block, you will be able to understand :

- Creation of Java
- Discuss Java Technology, Java Programming language and Java platform
- Define important terms of Java
- Compare Java with C++
- Garbage collection
- Java program and provide comments
- Tokens, data types in Java
- Declaring of a variable, Java coding conventions

Block Structure :

Unit 01

INTRODUCTION TO JAVA

UNIT STRUCTURE

1.0 Learning Objectives :

After learning this unit, you will be able to understand :

- Creation of Java
- Discuss Java Technology, Java Programming language and Java platform
- Define important terms of Java
- Compare Java with C++
- Describe garbage collection
- Illustrate Java program and provide comments

1.1 Introduction :

The current programming problems are complex as beyond a certain size, structured programming cannot manage complexity. Using the concept of object oriented programming, complex programs can be organized using classes, inheritance and polymorphism. C++ was one of the popular programming languages for Object Oriented Programming.

As speedy growth of Internet uses, Industry required a language that truly run on any operating system (Platform Independent). Java is the only language at that time that provide platform independent concept. As Java provide the feature of platform independent concept it widely used in electronic machine like washing machine, microwave oven etc. Using Java language we can develop the software that either run on standalone computer or on Web browser. In this unit, we'll discuss about the basics of Java Language.

1.2 The Creation of Java :

❖ **History of Java :**

James Gosling initiated the Java language project in June 1991 for use in one of his many set–top box projects. The language, initially called Oak after an oak tree that stood outside Gosling's office, also went by the name Green and ended up later renamed as Java, from a list of random words.

Sun released the first public implementation as Java 1.0 in 1995. It promised Write Once, Run Anywhere (WORA), providing no–cost run–times on popular platforms.

On 13 November 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).

On 8 May 2007, Sun finished the process, making all of Java's core code free and open–source, aside from a small portion of code to which Sun did not hold the copyright.

Figure 1.1 Java Logo

❑ **Check Your Progress – 1 :**

1. Who initiated Java language ?
2. What do the terms WORA and GPL mean ?

....

....

....

....

....

1.3 The Java Technology :

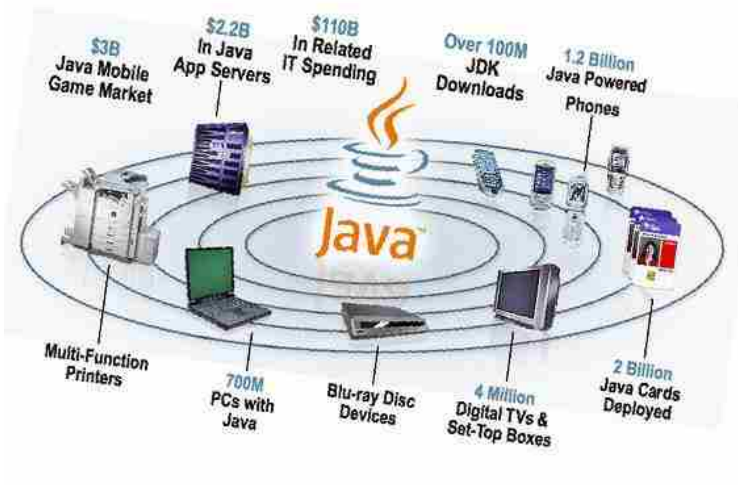

$3B Java Mobile Game Market
$2.2B In Java App Servers
$110B In Related IT Spending
Over 100M JDK Downloads
1.2 Billion Java Powered Phones
Java
Multi-Function Printers
700M PCs with Java
Blu-ray Disc Devices
4 Million Digital TVs & Set-Top Boxes
2 Billion Java Cards Deployed

Figure 1.2 The Java Technology

The Java technology is a programming language and a platform. Let us discuss Java as a programming language and a platform both.

❖ **Java Programming Language :**

Java is simple, object–oriented, multi–threaded, robust, portable and dynamic programming language. It is an object–oriented language similar to C++ but simplified to eliminate language features which cause common programming errors.

The Java source code files (files with .java extension) are compiled into a format called byte code (files with a .class extension) which can then be executed by a Java interpreter. The compiled Java code can run on most computers because Java interpreters and runtime environments which are known as Java Virtual Machines exist for most operating systems including UNIX, Macintosh OS and Windows.

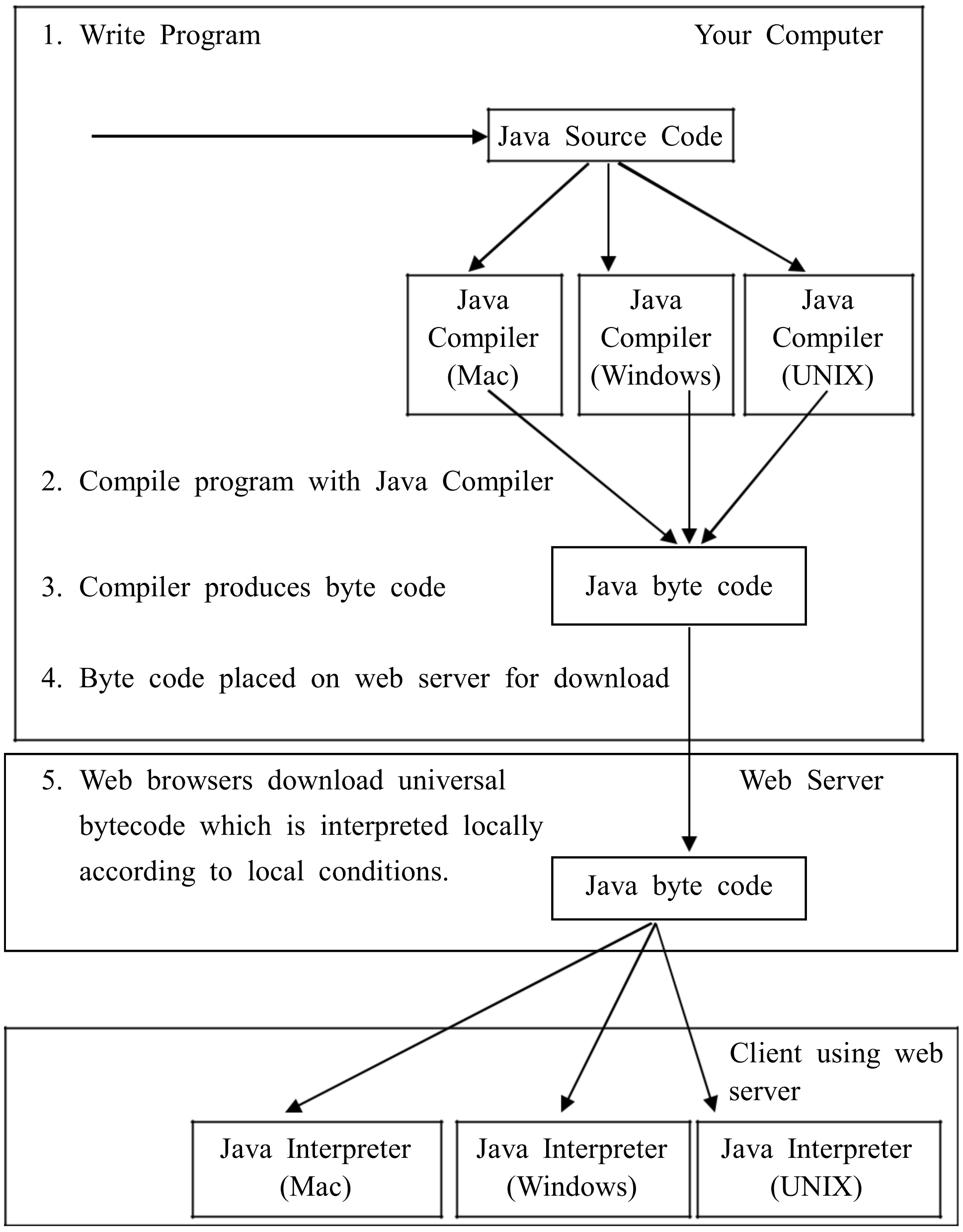

1. Write Program
Your Computer
Java Source Code
Java Compiler (Mac)
Java Compiler (Windows)
Java Compiler (UNIX)
2. Compile program with Java Compiler
3. Compiler produces byte code
Java byte code
4. Byte code placed on web server for download
5. Web browsers download universal bytecode which is interpreted locally according to local conditions.
Web Server
Java byte code
Client using web server
Java Interpreter (Mac)
Java Interpreter (Windows)
Java Interpreter (UNIX)

Figure 1.3 Java Programming Environment

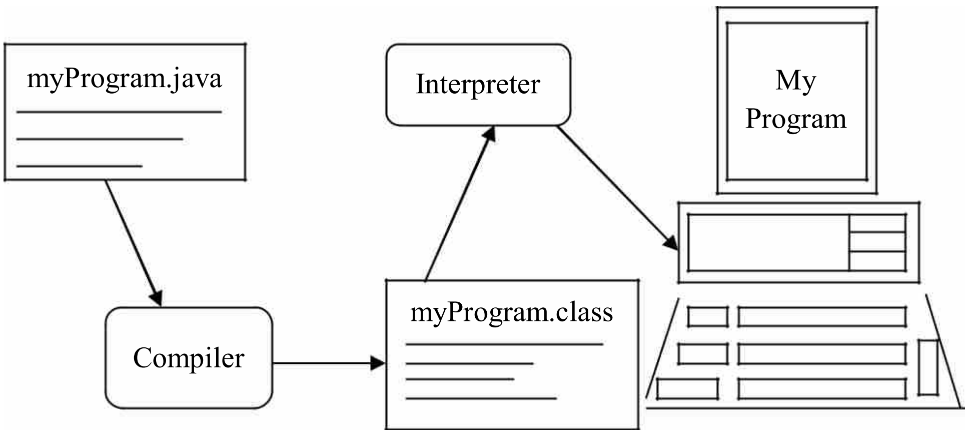


Figure 1.4 Execution of Java Program

❖ **The Java Platform :**

A platform can be defined as the hardware or software environment in which a program runs. Most of the platforms can be described as a combination of the operating system and hardware. The Java platform differs from most of the other platforms. It is a software only platform which runs on the top of other hardware based platforms. The Java platform has two components :

The Java Virtual Machine (JVM)

The Java Application Programming Interface (Java API)

The JVM is the base for the Java platform and is ported onto various hardware based platforms.

The Java API is a large collection of ready–made software components which provide many useful capabilities, such as Graphical User Interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as packages

❑ **Check Your Progress – 2 :**

1. What do you mean by platform ?
2. Name the two components of Java platform.

....

....

....

....

....

1.4 Features of Java :

Although the main features of Java are its portability and security. Apart from this, certain other features are also there which are discussed below :

1. **Simple –** Java is primarily derivative of C++. It omits rarely used and confusing features of C++ such as pre–processor, operator overloading, multiple inheritance etc.
2. **Object Oriented –** Java is an object oriented programming language, in which data is treated as objects to which methods are applied. Its basic execution unit is the class.

3. **Robust –** Java is strongly typed language. The type checking is carried out at both compile and runtime with every data structure clearly defined and typed. Java supports automatic garbage collection which manages memory by preventing memory leaks.

4. **Architecture Independent –** The Java compiler compiles source code and generates bytecode which is intermediate between source and machine code. These machine codes are neutral and have nothing to do with particular computer architecture. The Java Virtual Machine (JVM) coverts the bytecode into native code for a particular processor.

5. **Portable –** The interpreter for the Java Virtual Machine can be ported to any computer hardware/operating system, so that all the codes compiled for it will run on that system. This forms the basis for Java's portability.

6. **Multithreaded –** Multithreading helps to overcome the performance problems caused due to interpreted code as compared to the main code. Since an executing program hardly ever uses CPU cycles 100 percent of the time, Java uses the idle time to perform the necessary garbage cleanup and general system maintenance that renders.

7. **High Performance –** With the use of Just–In–Time compilers Java enables high performance.

8. **Distributed –** Java is designed for the distributed environment of the internet.

9. **Dynamic –** Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run–time information that can be used to verify and resolve accesses to objects on run–time.

10. **Secure –** With Java's secure feature it enables to develop virus–free, tamper–free systems. Authentication techniques are based on public–key encryption.

❑ **Check Your Progress – 3 :**

1. Explain the multi–threaded and dynamic feature of Java.
2. Write a note on portability and security feature of Java.

....

....

....

....

....

1.5 Comparison of Java with C++ :

Java is somewhat similar to C++. However, Java is not a superset or subset of C++; it can be seen as a derivation with many modifications and extensions. The given table displays the difference between them :

Table 1.1 : C++ Vs. Java

C++	JAVA
Hybrid between procedural and object–oriented language	Purely object–oriented language

No Automatic garbage collection	Automatic garbage collection
Programs are compiled	Programs are compiled and interpreted
Architecture specific	Architecture neutral
Platform Dependant (Programme can run on same Operating Systems on which it built)	Platform Independent (Programme can run on any Operating System)
Supports multiple inheritance	No multiple inheritance
Supports operator Overloading	Does not support operator overloading
Templates as parameterized type	No parameterized type
Supports Pointers with dereferencing (* or –>) and address (&) operators.	No explicit pointer manipulation and no pointer arithmetic
Strings are null–terminated character arrays	Strings are objects
Main function can return a value	Main method cannot return a value

❑ **Check Your Progress – 4 :**

1. Compare Java with C++ with respect to garbage collection and inheritance.
2. Give the difference between Java and C++ with respect to strings and operator overloading.

....

....

....

....

....

1.6 Garbage Collection :

Figure 1.5 Garbage Collection

In C++ language, the dynamically allocated objects are manually released by the use of a delete operator. In Java, the deallocation process is automatically done and the technique to do that is called garbage collection.

When no references to an object exist, that object is no longer needed and the memory occupied by the object can be reclaimed. The explicit destruction of objects is not done as in C++.

Garbage collection occurs at irregular intervals during the execution of program. It will not occur simply because one or more objects exist which are no longer needed.

Different Java run–time implementations take varying approaches to garbage collection, so it is recommended that one should not think about it while writing programs.

❑ **Check Your Progress – 5 :**

1. Explain the deallocation process in Java and C++.
2. What is the occurrence of garbage collection ?

....

....

....

....

....

1.7 Creating a Java Program :

The execution of Java Program is divided into several steps. These steps are discussed below :

a. Writing the Code : The steps performed while writing the code are :

   1. Use any text editor
   2. Save the program as.java
   3. Name the program the same as the class containing main method

Compiling the Code : The process of compiling the code involves the given steps :

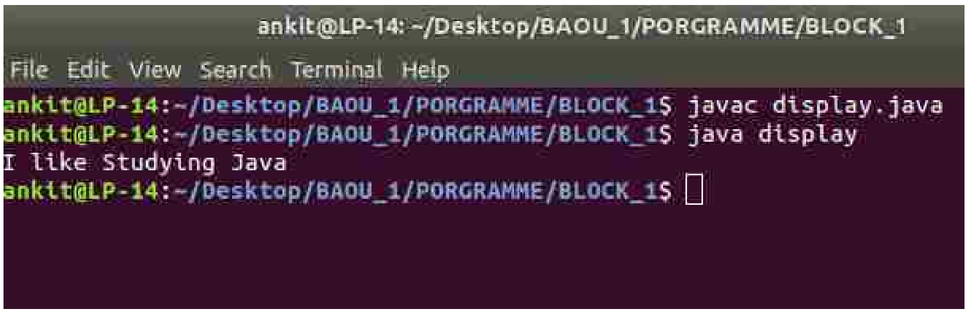
- Use the Command prompt
- Javac programname.java
- Creates porgramname.class file

b. Interpreting the ByteCodes : The ByteCodes are interpreted by following the given steps :

- Use the command prompt
- .java programname

After studying about the steps of creating a Java Program, let us write a small and simple program to display "I like Studying Java" on the screen.

```
// Program to display message on screen
class display
{
 public static void main (String [ ] args)
 {
  System.out.println("I like Studying Java");
  //Displays the string
 }
}
```



```
ankit@LP-14: ~/Desktop/BAOU_1/PORGRAMME/BLOCK_1
File Edit View Search Terminal Help
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_1$ javac display.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_1$ java display
I like Studying Java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_1$
```

Figure 1.6 Output of Program

Now let us try to understand the above program. The first point of execution of any Java program is its main method. This main method has to be defined within a class.

In order to run an application with the Java interpreter, the name of the class has to be specified which has to be executed. The interpreter invokes the main method defined within that class.

The method signature for the main method contains three modifiers :

- Public indicates that the main method can be accessed outside the class in which it is declared.
- Static indicates that the main method can be invoked without creating an instance of the class.
- Void indicates that the main method doesn't return any value.

The main method accepts a single argument, that is, an array of elements of class String and arg receives command line arguments. The next line is System.out.println ("I like Studying Java"), this statement displays the string "I like Studying Java".

The output gets displayed by println ( ) method. The println ( ) displays the string which is passed to it. This method can also be used to display other types of information also. It begins with System.out; here System is class and out is the output stream which is connected to the console. Thus, System.out is an object which encapsulates console output.

❖ **Providing Comments :**

Comments are used to provide brief documentation to the program. There are two most commonly used methods of providing comments to Java program :

a. **Single Line Comment –** The single line comment is used to provide comments of one line. It starts with // and ends with new line character. For example,

   // this is my first Java Program

b. **Multiline Comment –** When the comment to be included is of multiple lines, then multiline comment is used. It starts with /* and ends with */. For example,

   /* This is

   My first

   Java Program */

❑ **Check Your Progress – 6 :**

1. Explain the steps involved in the execution of Java program.

   ....

   ....

   ....

   ....

   ....

2. Which Company released the first public implementation as Java 1.0 in 1995.

   (A) Sun (B) Microsoft (C) Oracle (D) Wipro

3. ________ is an extension of java code file.

   (A) .txt (B) .cpp (C) .java (D) .javac

4. JVM stand for ________.

   (A) Java Virtual Machine (B) JDK Vision Mission

   (C) Java Vision Machine (D) Java Virtual Mission

5. The ________ is a large collection of ready–made software components.

   (A) Java (B) Java API (C) JVM (D) JDK

6. The deallocation process is automatically done by ________.

   (A) Garbage Collector (B) Pointer

   (C) Memory (D) RAM

7. Java programme is compiled by using ______ command.

   (A) java (B) jdk (C) cc (D) javac

8. ________ is used as a single line comment.

   (A) /* (B) // (C) */ (D) **

9. Java is simple, object–oriented, multi–threaded, robust, portable and dynamic programming language.

   (A) False (B) True

10. ".class" is an extension of byte code file

    (A) False (B) True

11. Java is an Procedure oriented programming language

    (A) False (B) True

1.8 Let Us Sum Up :

This Unit gives lot of basic learning right from History of Java that James Gosling initiated the Java language project in June 1991 for use in one of his many set–top box projects. The language, initially called Oak after an oak tree that stood outside Gosling's office, also went by the name Green and ended up later renamed as Java.

As the current programming problems are complex, as beyond a certain size, structured programming cannot manage complexity. Using the concept of object oriented programming, complex programs can be organized using classes, inheritance and polymorphism. C++ was one of the popular programming languages for Object Oriented Programming.

Java is simple, object–oriented, multi–threaded, robust, portable and dynamic programming language. It is an object–oriented language similar to C++ but simplified to eliminate language features which cause common programming errors.

A platform can be defined as the hardware or software environment in which a program runs. Most of the platforms can be described as a combination of the operating system and hardware. The Java platform has two components :

- The Java Virtual Machine (JVM)
- The Java Application Programming Interface (Java API)

Although the main features of Java are its portability and security. Apart from this, certain other features are also there which are discussed below :

1. Simple and
2. Object Oriented

About Garbage collection in C++ language, the dynamically allocated objects are manually released by the use of a delete operator. In Java, the deallocation process is automatically done and the technique to do that is called garbage collection.

Creating and executing a Java Program is divided into several steps. These steps are discussed below :

a. Writing the Code : The steps performed while writing the code are :
   - Use any text editor
   - Save the program as .java
   - Name the program the same as the class containing main
b. Compiling the Code : The process of compiling the code involves the given steps–
   - Use the Command prompt
   - .javac programname.java
   - Creates programname.class file
c. Interpreting the ByteCodes : The ByteCodes are interpreted by following the given steps :
   - Use the command prompt
   - .java programname

Comments are used to provide brief documentation to the program. There are two most commonly used methods of providing comments to Java program

Single Line Comment : The single line comment is used to provide comments of one line. It starts with // and ends with new line character. Multiline Comment : When the comment to be included is of multiple lines, then multiline comment is used. It starts with /* and ends with */.

1.9 Suggested Answer for Check Your Progress :

❑ **Check Your Progress 1 :**

See Section 1.2

❑ **Check Your Progress 2 :**

See Section 1.3

- **Check Your Progress 3 :**

  See Section 1.4
- **Check Your Progress 4 :**

  See Section 1.5
- **Check Your Progress 5 :**

  See Section 1.6
- **Check Your Progress 6 :**

  1 : See Section 1.7

2 : A	**3 :** C	**4 :** A	**5 :** D	**6 :** A
7 : D	**8 :** B	**9 :** D	**10 :** B	**11 :** A

1.10 Glossary :

1. **Java Virtual Machines** – The compiled Java code can run on most computers because Java interpreters and runtime environments which are known as Java Virtual Machines
2. **Object Oriented** – Data is treated as objects to which methods are applied. Its basic execution unit is the class.
3. **Portable** – The interpreter for the Java Virtual Machine can be ported to any computer hardware/operating system, so that all the codes compiled for it will run on that system. This forms the basis for Java's portability.
4. **Dynamic** – Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run–time information that can be used to verify and resolve accesses to objects on run–time.
5. **Java Platform** – The hardware or software environment in which a program runs.

1.11 Assignment :

Discuss the function of modifiers in the method signature of main method.

1.12 Activities :

Find the new terminologies and write down the meaning and importance of each.

1.13 Case Study :

1. Write a program to display a message on screen and explain the execution of the program
2. Write a program to display "Well come to BAOU" on screen.
3. Write a program to display your name on screen.

1.14 Further Reading :

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000
2. Java 2, The Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999

3. Programming with Java, Ed. 2,E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000
4. The Java Tutorial, Ed. 2, 2 volumes, Mary Campione and Kathy Walrath, Addison Wesley Longmans, 1998
5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele & Gilad Bracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000
6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition, 2000

Unit 02

PROGRAMMING CONCEPTS OF BASIC JAVA

UNIT STRUCTURE

2.0 Learning Objectives :

After learning this unit, you will be able to understand :

- Describe Tokens, data types in Java
- Declare a variable, Java coding conventions
- Define typecasting
- Explain constants

2.1 Introduction :

Java is an object–oriented programming language as it works totally on the concepts of class and object programming concepts. Object oriented programming (called OOP for short) is the concept, which got its way after the procedural programming languages which was developed in 1970's.

The programming language that existed before was more of process oriented and this gave the concept of small entities called objects which could be made and reused as and when the need arises.

OOP hails from the idea of objects. All the real life entities are basically nothing but objects. In any program, every code will have an object having few characteristics features called the properties of that particular object or entity and then it will always have few actions to be performed over those entities or objects termed as Methods or Functions. To work with OOP, you should be able to identify three key characteristics of objects :

- **The behaviour of object :** what can you do with this object, or what methods can you apply to it ?
- **The state of the object :** how does the object react when you apply those methods ?
- **The identity of the object :** how is the object distinguished from others that may have the same behaviour and state ?

2.2 Tokens :

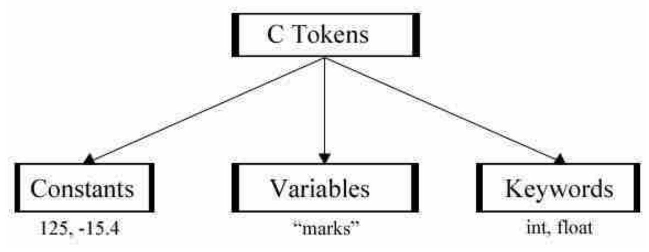


Figure 2.1 Tokens

The tokens of a language are the basic building blocks which can be put together to construct programs. A token can be a reserved word (such as int or while), an identifier (such as b or sum), a constant (such as 25 or "Alice in Wonderland"), a delimiter (such as {or ;) or an operator (such as + or =). These tokens are explained below :

Identifiers – Identifiers are those words, which help to identify an entity. It can be used for class names, method names and variable names. It can be represented using uppercase and lowercase characters, numbers, underscore and dollar sign characters.

For Example,

a. Class_Name

b. B5

c. $name

d. This_is_program

Literals/Constants – Literals or constants are those quantities whose value may not change during execution of program. Java support five types of Literals / constants.

- Integer Literals
- Float Literals
- Character Literals
- String Literals
- Boolean Literals

1. Integer Literals :

It store whole numbers only. Integer Literals supports three subtypes of Integer Literals.

- Decimal – it has value between 0 to 9
  (i.e – 20, 45, 99, 102)
- Octal – it has value between 0 to 7
  (i.e – 12, 45, 33)

- Hexadecimal – it has value between 0 to 9 and A to F (i.e – 0Xa2f, 0X3ba)

2. Float Literals :

It store fractional values. It store fractional values in either standard or scientific forms.

Standard Form – i.e – 3.14, 22.879, 436.9845

Scientific Form – i.e – 8.46 * 10^{13}, 69.44 * 1033

3. Character Literals :

Single Character is represent Character Literals. Some special character (Backslash character) is also consider as a Character Literals. The Backslash Character is explain in blow table.

Backslash Character	**Purpose**
\"	Double Quote
\'	Single Quote
\t	Tab
\\	Backslash
\n	New Line
\b	Backspace
\f	Form Feed
\r	Carriage Return

4. String Literals :

A collection of character within the double quotes is called String Literals.

5. Boolean Literals :

It has any one value out of two values like either True or False.

In Java, the keyword final is used to denote a constant. The value of final variable cannot change after it has been initialised.

For example,

final int x=0;

The above statement declares a final variable and initialises it, all at once.

While attempting to assign a value to variable x would result in compilation error.

Keywords – Keywords are also called reserved words. These are those words whose meaning is already explained to the compiler, so that when they are used in the program the compiler never generates error. There are mainly 49 keywords used in Java language. These keywords are given in the following table :

Table 2.1 : Keywords

abstract	do	import	public	transient
boolean	double	instance of	return	try
break	else	int	short	void
byte	extends	interface	static	volatile
case	final	long	super	while
catch	finally	native	switch	
char	float	new	synchronized	
class	for	package	this	
continue	if	private	throw	
default	implements	protected	throws	

In addition to the above keywords, true, false and null are also the reserved words whose values are defined by Java.

❑ **Check Your Progress – 1 :**

1. Write a note on identifiers.
2. Explain constants

....

....

....

....

....

2.3 Data Types in Java :

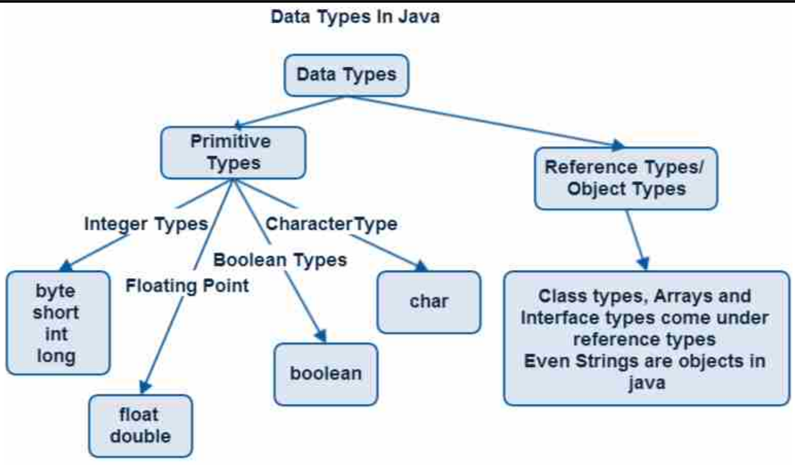


Figure 2.2 Data Types in Java

- Java is a strongly typed language. Every variable, expression has a type and these types are strictly checked. Unlike C, type checking is strictly enforced at run time.
- Impossible to typecast incompatible types.

  In Java, a floating point value can be assigned to an integer.

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the memory. Therefore, by assigning different data types to variables, you can store Int, Float, Char, Boolean (any of the eight primitive data types in these variables.

There are two data types available in Java :

1. Primitive Data Types
2. Reference/Object Data Types

❖ **Primitive Data Types :**

There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a key word. Let us now look into detail about the eight primitive data types.

❖ **Byte :**

- Byte data type is an 8–bit signed.
- Minimum value is –128
- Maximum value is 127
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.
- Example : byte a = 100 , byte b = –50

❖ **Short :**

- Short data type is a 16–bit signed.
- Minimum value is –32,768
- Maximum value is 32,767
- Short data type can also be used to save memory as byte data type.
- Default value is 0.
- Example : short s= 10000 , short r = –20000

❖ **int :**

- int data type is a 32–bit signed.
- Minimum value is 2,147,483,648.
- Maximum value is 2,147,483,647.
- int is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0.
- Example : int a = 200, int b = –400

❖ **Long :**

- Long data type is a 64–bit signed.
- Minimum value is –9,223,372,036,854,775,808.
- Maximum value is 9,223,372,036,854,775,807.
- This type is used when a wider range than int is needed.
- Default value is 0L.
- Example : int a = 200L, int b = –400L

❖ **Float :**

- Float data type is a 32 bit floating point numbers.
- Float is mainly used to save memory in large arrays of floating point numbers.
- Default value is 0.0f.
- Float data type is never used for precise values such as currency.
- Example : float f1 = 134.5f

❖ **Double :**

- Double data type is a double–precision 64–bit floating point.
- This data type is generally used as the default data type for decimal values.
- Default value is 0.0d.
- Example : double d1 = 113.4

❖ **Boolean :**

- boolean data type represents one bit of information.
- There are only two possible values : true and false.
- This data type is used for simple flags that track true/false conditions.
- Default value is false.
- Example : boolean b = false

❖ **Char :**

- Char data type is a single 16–bit Unicode character.
- Minimum value is '\u0000' (or 0).
- Maximum value is '\uffff' (or 65,535 inclusive).
- Char data type is used to store any character.
- Example : char letter ='C'

❖ **Reference Data Types :**

- Hold the reference of dynamically created objects which are in the heap memory
- Can hold three types of values :
- Class type : Points to an object / instance of a class
- Interface type : Points to an object, which is implementing the corresponding interface
- Array type : Points to an array object or "null"
- Difference between Primitive & Reference data types :
- Primitive data types hold values themselves
- Reference data types hold reference to objects, i.e. they are not objects, but reference or pointers to objects
- Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, employee, puppy etc.

- Class objects and various types of array variables come under reference data type.
- Default value of any reference variable is null.
- A reference variable can be used to refer to any object of the declared type or any compatible type.
- Example : Animal animal = new Animal("elephant").

❑ **Check Your Progress – 2 :**

1. Explain float data type.
2. Write a note on reference data type.

....

....

....

....

....

2.4 Declaring a Variable :

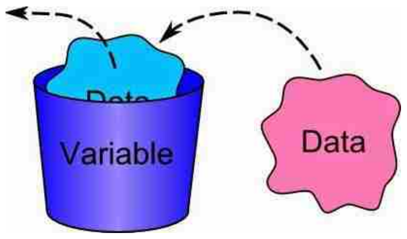


Figure 2.3 Declaring a Variable

In Java, variables are those quantities whose value may change during execution of program. These variables should be initialised before they are used. The syntax of variable declaration is given below :

❖ **Syntax :**

Type identifier [=value] [, identifier [=value]…];

Where,

Type is one of Java's atomic types, i.e., the name of class or interfaces.

A variable declared in java need to followed naming convention. The rules for declaring the variable are as follow :

- The first character in a variable name should not be a digit.
- A variable name may consist of alphabets, digits, the underscore character and the dollar character.
- A keyword should not be used.
- White spaces are not allowed.
- A variable name can be of any length.

For example, the given statements display the method of variable declaration.

```
int a, b, c;            //will declare three variables a, b and c of integer
                        types
int x=5, y=10, z=25;    //will declare 3 variables x, y and z with 5, 10
                        and 25 values.
```

```
double pi=3.14;     //will declare an approximate value of pi
char x= 'a';        //will declare variable x with character value 'a'
```

❑ **Check Your Progress – 3 :**

1. What is a variable ?
2. Write the syntax of declaring variables.

....

....

....

....

....

2.5 Java Coding Conventions :

Reducing the cost of software maintenance is the significant reason for following coding conventions. In their introduction to code conventions for the Java Programming Language, Sun Microsystems provides the rationale.

Code conventions are important to programmers for a various reasons :

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the same programmer.
- Code conventions improve the readability of the software, allowing software engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product.
- Coding conventions allows simple scripts or programs whose job is to process source code for some purpose other than compiling it into an Executable. It is common practice to count the software size (Source lines of code) to track current project progress or establish a baseline for future project estimates.
- Consistent coding standards can, in turn, make the measurements more consistent. Special tags within source code comments are often used to process documentation.

❑ **Check Your Progress – 4 :**

1. Why code conventions are vital to programmers ?
2. Explain the use of special tags in source code comments.

....

....

....

....

....

2.6 Typecasting :

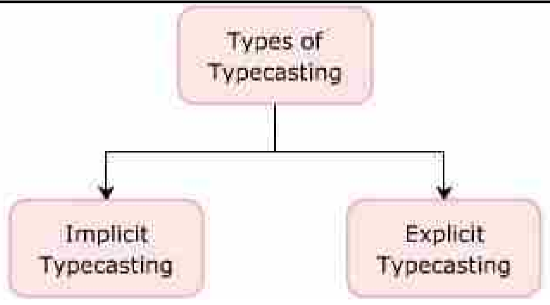


Figure 2.4 Typecasting

The conversion of one data type to another data type is called typecasting or type conversion.

There are mainly two types of conversions; these are implicit or automatic and explicit conversions.

a. Automatic Conversion :

The type conversions are automatically performed when the type of the expression on the right hand side of an assignment operation can be safely promoted to the type of the variable on the left hand side of the assignment.

Thus, the values can be safely assigned as :

byte–>short–>int–>long–>float–>double

The extra storage associated with long integer as shown in the above example will be padded with extra zeroes.

For Example,

byte b1 = 30;

int n = b1;

Here byte is lower precision type so it can be automatically convert in to Integer.

b. Explicit Conversion :

If we want to assign the value of long to an integer then an integer variable will require more storage and may result in loss of data. To force such conversion, an explicit conversion is performed and the process is called explicit typecasting.

For example,

If

int x;

long y;

Then,

x= (int) y;

The above statement tells the compiler that the type of variable y must be temporarily changed to an int when the given assignment statement is processed. Thus, the cast only lasts for the duration of the assignment.

So, the syntax of explicit typecasting can be given as :

(T) N

Here, T is the name of a numeric type and N is a data item of another numeric type. The result is of type T

❑ **Check Your Progress – 5 :**

1. What do you mean by the term typecasting ?
2. Explain explicit conversion with suitable example.

....

....

....

....

....

2.7 Constants :

Java is a programming language used to create programs that can run on a variety of Operating Systems. A Java constant is a variable with a pre–defined value. Although Java does not have a constant type, you can effectively obtain the same effect with a final variable. This enables you to have a control of what is constant and what is not.

You can have a constant effect by declaring and initializing public, static and final variables. The static modifier makes the variable obtainable without loading an occurrence of the class where it is defined.

Once you have 34 initialized the constant variables, their value cannot be changed anymore. After initializing, you can gain access to the constant value with the variable's name and the name of its class with a period.

❖ **Standard Naming Convention :**

In declaring Java constant variables, you should declare the variable names in ALL CAPS (notice each letter of the variable name above). The words in Java constants are typically separated with underscores (as in the example above). This format indicates that these values are constants. It will be easier for an individual to read a code if this standard naming convention is followed.

❑ **Check Your Progress – 6 :**

1. How can a constant effect be achieved ?
2. Illustrate an example of Java constant.

....

....

....

....

....

3. ________ is a basic buliding block of java programs.

   (A) Token (B) Variable (C) Method (D) Class

4. ________ is used to identify the entity.

   (A) Java (B) Identifiers (C) Token (D) Kewords

5. ________ literals has value between 0 to 7.

   (A) Float (B) Octal Integer (C) Double (D) Boolean

6. The maximum value of Byte data type is ________.

   (A) 128 (B) 127 (C) –127 (D) 100

7. The default value of Int data type is ________.

   (A) 0 (B) 1 (C) –0 (D) Garbage

8. The default value of boolean data type is ________.

   (A) false (B) true (C) yes (D) no

9. Default value of any reference variable is ________.

   (A) null (B) NaN (C) 0 (D) Garbage

10. Hexadecimal Integer literals has value between 0 to 9 and A to F

    (A) True (B) False

11. Reference data types are predefined by the language and named by a key word.

    (A) True (B) False

12. Float data type is never used for precise values such as currency.

    (A) True (B) False

2.8 Let Us Sum Up :

This unit deals with several important basic aspects of Java Programming. One of them is the tokens of a language which are the basic building blocks and can be put together to construct programs. A token can be a reserved word (such as int or while), an identifier (such as b or sum), a constant (such as 25 or "Alice in Wonderland"), a delimiter (such as { or ;) or an operator (such as + or =). These tokens are explained below :

Identifiers – Identifiers are those words, which help to identify an entity. It can be used for class names, method names and variable names. It can be represented using uppercase and lowercase characters, numbers, underscore and dollar sign characters as a) Java is a strongly typed language. Every variable, expression has a type and these types are strictly checked. Unlike C, type checking is strictly enforced at run time. 2) Impossible to typecast incompatible types. In Java, a floating point value can be assigned to an integer. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the memory. Therefore, by assigning different data types to variables, you can store int, float, char, boolean (any of the eight primitive data types in these variables. There are two data types available in Java : 1. Primitive Data Types 2.Reference/Object Data Types In Java, variables are those quantities whose value may change during execution of program. These variables should be initialized before they are used. The syntax of variable declaration is given below :

Syntax – Type identifier [=value] [, identifier [=value]…];

Reducing the cost of software maintenance is the significant reason for following coding conventions. In their introduction to code conventions for the Java Programming Language, Sun Microsystems provides the rationale. Code conventions are important to programmers for a various reasons. Further it is understood that the conversion of one data type to another data type is called typecasting or type conversion. There are mainly two types of conversions; these are implicit or automatic and explicit conversions.

Java is a programming language used to create programs that can run on a variety of Operating Systems. A Java constant is a variable with a pre–defined value. Although Java does not have a constant type, you can effectively obtain the same effect with a final variable. This enables you to have a control of what is constant and what is not.

You can have a constant effect by declaring and initializing public, static and final variables. The static modifier makes the variable obtainable without loading an occurrence of the class where it is defined. Once you have to initialize the constant variables; their value cannot be changed anymore. After initializing, you can gain access to the constant value with the variable's name and the name of its class with a period. We learned that there is Standard Naming Convention. In declaring Java constant variables, you should declare the variable names in ALL CAPS (notice each letter of the variable name above). The words in Java constants are typically separated with underscores (as in the example above). This format indicates that these values are constants. It will be easier for an individual to read a code if this standard naming convention is followed.

2.9 Suggested Answer for Check Your Progress :

❑ **Check Your Progress 1 :**

See Section 2.2

❑ **Check Your Progress 2 :**

See Section 2.3

❑ **Check Your Progress 3 :**

See Section 2.4

❑ **Check Your Progress 4 :**

See Section 2.5

❑ **Check Your Progress 5 :**

See Section 2.6

❑ **Check Your Progress 6 :**

1 : See Section 2.7 **2 :** See Section 2.7

3 : C	**4 :** B	**5 :** B	**6 :** A	**7 :** A
8 : B	**9 :** A	**10 :** A	**11 :** A	**12 :** A

2.10 Glossary :

1. **Tokens** – The tokens of a language are the basic building blocks which can be put together to construct programs.
2. **Identifiers** – Identifiers are those words, which help to identify an entity. It can be used for class names, method names and variable names.
3. **Literals/Constants** – Literals or constants are those quantities whose value may not change during execution of program.
4. **Typecasting** – conversion of one data type to another data type is called typecasting or type conversion.

2.11 Assignment :

1. Discuss the various data type available under Java.
2. Discuss the type conversion.

2.12 Activities :

1. Write the steps to save a file, compile and run program.
2. Explain class, object, methods and instant variables
3. Write a program that convert meter value in to kilometres.
4. Write a program that calculate the area of square.
5. Write a program that calculate the area of triangle.

2.13 Case Study :

1. Write a program that show the use of datatypes available in java.

2.14 Further Reading :

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000
2. Java 2, The Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999
3. Programming with Java, Ed. 2,E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000
4. The Java Tutorial, Ed. 2, 2 volumes, Mary Campione and Kathy Walrath, Addison Wesley Longmans, 1998
5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele & Gilad Bracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000
6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition, 2000

Unit 03

OPERATORS AND PRECEDENCE

UNIT STRUCTURE

3.0 Learning Objectives :

After learning this unit, you will be able to understand :

- Arithmetic Operator
- Assignment Operator
- Bitwise Operator
- Relation Operator
- Logical Operator
- Ternary Operator
- Operator Precedence

3.1 Introduction :

An Operator is a symbol that allows a us to perform arithmetic or logical operations on data. It operate on operands and cause changes in the operand value. Java provides a rich set of operators for manipulating programs. Commonly operator performs a function on one, two or three operands.

Unary operator perform operation based on one operand. The ++ or – – are the Unary operators. Binary operator perform operation based on two operands. The +, <, = are Binary operators. An operator that required three operands are called Ternary operator. The "expression1 ? expression2 : expression3" is an example of Ternary operator.

In Java, Operators are divided in to six categories. :

- Arithmetic Operators
- Assignment Operators
- Bitwise Operators
- Relational Operators
- Logical Operators
- Ternary Operator

3.2 Arithmetic Operator :

The arithmetic operators are used to perform arithmetical operations. For example, addition, subtraction, multiplication, division and modulo.

These arithmetic operators can be unary or binary type. If the operator is specified with two operands then it is called binary operator and if the operators are used with single operand then they are called unary operators. The given table gives a brief description of binary operators :

Here the values of Variables A = 10 and B = 20

Table 3.1 : Binary Operators (Arithmetic Operator)

Operator	Description	Example
+	Addition – Adds values of either side of the operator	A + B will give 30
–	Subtraction – Subtracts right hand operand from left hand operand	A – B will give –10
*	Multiplication – Multiplies values on either side of the operator	A * B will give 200
/	Division – Divides left hand operand by right hand operand	B / A will give 2
%	Modulus – Divides left hand operand by right hand operand and returns remainder	B % A will give 0

❑ **Check Your Progress – 1 :**

1. Write a note on Arithmetic operators.

....

....

Following program shows the use of Arithmetic Operator

```
class ArithmaticDemo
{
   public static void main(String args[])
   {
      int a = 40;
      int b = 20;
```

```
        //Addition (+) operator
        System.out.println("Addition of a and b = " + (a + b));

        //Subtraction (-) operator
        System.out.println("Subtraction of a and b = " + (a - b));
        //Multiplication (*) operator
        System.out.println("Multiplication of a and b = " + (a * b));

        //Division (/) operator
        System.out.println("Division of a and b = " + (a / b));

        //Modulo (%) operator
        System.out.println("Modulo of a and b = " + (a % b));

    }
}
```

OUTPUT :

```
Addition of a and b = 60
Subtraction of a and b = 20
Multiplication of a and b = 800
Division of a and b = 2
Modulo of a and b = 0
```

3.3 Increment / Decrement Operator :

The binary forms of ++ and – – are unary operator. It is also know as increment / decrement operator. Each of these operator has unary versions that perform the following operations :

Table 3.2 : Increment / Decrement Operators

Operator	Description	Example
+ +	Increment by one value	+ +a, b+ +
– –	Decrement by one value	– –a, b– –

The unary operator also known as increment and decrement operators. The Increment and decrement operators can be used in two ways, either before or after an operand. Depending on the placement of these operators, it can be called as prefix or postfix operation. In the prefix notations, the value of the operands is incremented or decremented before assigning it to another variable, while in the postfix notation, the value of the operands is incremented or decremented after assigning it to another variable.

There are two types of increment and decrement operators :

1. Pre–increment/Decrement
2. Post–increment/Decrement

1. **Pre–Increment/Decrement –** The pre–increment/decrement operator increases/decreases the value of a variable first and then evaluates the expression.

   For example,

   If a = 2, then b = ++a will first increase the value of a that is make it 3 and then assign it to variable a.

   So, the final value of b becomes 3.

   Similarly, if a = 2 then b = – –a will first decrease the value of variable a by 1 and then assign it to variable b.

2. **Post–Increment/Decrement –** The post–increment/decrement operator first assigns the value to the variable and then increases/decreases it.

   For example, If a = 2; And b = a++;

   Then, first the value of a is assigned to b and then it is incremented/ decremented by 1.

   As in the above example, first value of a is assigned to b and then it gets increased. So, the value of b becomes 2 and a becomes 3.

   Similarly, if a = 2 and b = a– –, then value of b becomes 2 and a becomes 1.

   Following program shows the use of Unary Operator ++ :

```
class UnaryDemo
{
   public static void main(String args[])
   {
      int i, j, k;
      i = 10;
      j = i++;

      //prefix increment
      System.out.println("i = " + i);
      System.out.println("j = " + j);

      i = 10;
      j = 0;
      j = ++j;

      //postfix increment
      System.out.println("i = " + i);
      System.out.println("j = " + j);
   }
}
```

OUTPUT

```
i = 11
j = 11

i = 11
j = 11
```

❑ **Check Your Progress – 2 :**

1. Write a note on increment and decrement operators.

....

....

3.4 Assignment Operator :

Assignment operators are used to assign a value to operand (variable). Assignment Operator is a Binary operator. General form of operators is as follow :

varname = expr;

Where, varname is variable name and expr is an expression.

Table 3.3 : Assignment Operators

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assigne value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
– =	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C –= A is equivalent to C = C – A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2

>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

3.5 Bitwise Operator :

The bitwise operator is used when the manipulation is to be done bit by bit, i.e., in terms of 0's and 1's. These are faster in execution than arithmetic operators. The given table displays the bitwise operators along with their meanings :

Table 3.4 : Bitwise Operators and their meanings

Operators	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	One's complement
<<	Left Shift
>>	Right Shift
>>>	Right Shift with zero fill

❖ **Bitwise Not (~)**

The bitwise operators perform operations on integer value. In the above table, the ~ operator is unary operator and the rest of the operators are binary operators.

The ~ (one's complement) operator inverts the bits which make up the integer value, that is, 0 bits becomes 1 and 1 bits becomes 0.

For example, if ~3 is written, then it will give the value of ~4. The same can be calculated as follows :

The Numerical value 3 can be represented as binary integer value–

00000000 00000000 00000000 00000011

Inverting each bits would give

11111111 11111111 11111111 11111100

It is same as bit pattern for –4 as an int value.

The given table shows the operations performed at bit level :

Table 3.5 : Operations Performed at Bit Level

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give –60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>>	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>> 2 will give 15 which is 0000 1111

Now, based on the above table and explanations, let us take an example to understand the use of these bitwise operators. Consider the expressions, 63 & 252, 63 | 252 and 63 ^ 252

❖ **Bitwise AND (&)**

First of all we will calculate 63 & 252. To do the same, first represent the 63 and 252 values in their bit patterns.

```
00000000    00000000    00000000    00111111    = 63
00000000    00000000    00000000    11111100    = 252
```

As you know that and returns a 1 bit if and only if the corresponding bit from each operand is 1, we calculate 63 and 252 to be 60 as follows :

```
00000000    00000000    00000000    00111111    = 63
00000000    00000000    00000000    11111100    = 252
00000000    00000000    00000000    00111100    = 60
```

❖ **Bitwise OR (|)**

Now we will calculate 63 | 252. To do the same, first represent the 63 and 252 values in their bit patterns.

```
00000000    00000000    00000000    00111111    = 63
00000000    00000000    00000000    11111100    = 252
```

As you know that for Bitwise OR, Operator returns a 0 bit if, and only if, the corresponding bit form of each operand is 0. Else it returns 1. We calculate 63 OR 252 to be 255 as follows :

```
00000000    00000000    00000000    00111111    = 63
00000000    00000000    00000000    11111100    = 252
00000000    00000000    00000000    11111111    = 255
```

Bitwise XOR (|)

Now we will calculate 63 ^ 252. To do the same, first represent the 63 and 252 values in their bit patterns.

```
00000000    00000000    00000000    00111111    = 63
00000000    00000000    00000000    11111100    = 252
```

As you know that for Bitwise XOR, Operator returns a 0 bit if, and only if, the corresponding bit form of each operand match. Else it returns 1. We calculate 63 XOR 252 to be 195 as follows :

```
00000000    00000000    00000000    00111111    = 63
00000000    00000000    00000000    11111100    = 252
00000000    00000000    00000000    11000011    = 195
```

❖ **Shift Operators**

The shift operators work on the bit level. When the left operand is an int, only the last 5 bit of the right operand is used to perform the shift. This is due to the fact that an int is a 32 bit value and can only be shifted 0 through 31 times. Similarly, when the left operand is a long value, only the last 6 bits of the right operand are used to perform the shift, as long values are 64 bit values, they can only be shifted 0 through 63 times.

The << operator (left shift) causes the bits of the left operand to be shifted to the left based on the value of the right operand. The shifted right bits will be filled with 0 values.

The >> (right shift) operator causes the bits to the left operand to be shifted to the right, based on the value of the right operand. The bits that fill in the shifted left bits have the value of the leftmost bit (before the shift operation). This operator is also called signed shift as it preserves the sign (positive or negative) of the operand.

The >>> operator is similar to >> (Right shift) operator, except that the bits that fill in the shifted left bits have the value of 0. It is also called an unsigned shift as it does not preserve the sign of the operand.

❑ **Check Your Progress – 3 :**

1. Write a note on Bitwise operators.

   ....

   ....

3.6 Relation Operator :

The relational operators are used to compare two quantities. It either returns true or false value only.

For example,

num1>num2

Here, the value of num1 is checked with num2, if num1 is greater than num2 then a true value is returned else false. The syntax for declaring the relational operators is given below :

expr1 <relational operator> expr2

In the above syntax, expr1 and expr2 are the arithmetic expressions which can be variables, constants or both. When the arithmetic expressions are used on either side of a relational operator, the arithmetic expression gets evaluated first.

The given table shows the list of relational operators with their meaning :

Table 3.6 : Relational Operators

Operator	Description	Example
= =	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A = = B) is not true.
|=	Check if the value of the operands are equal or not, if values are not equal then condition becomes true.	(A |= B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not ture.
<	Checks if the value of left oeprand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

3.7 Logical Operator :

Logical operators are used to combine more than one condition to perform logical operation. There are 3 logical operators, the given table shows the list of these operators

Table 3.7 : Logical Operators

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then then condition becomes true.	(A && B) is false.
||	Called Logical OR operator. If any of the two operands are non zero then then condition becomes true.	(A || B) is true.
!	Called Logical NOT operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

The && and || are used to form compound conditions. For example, num1>num2 && num1 >num3

In the above expression, first the value of num1 and num2 will be compared (to the left side of && operator) then the values of num1 and num3 gets compared (to the right of &&) and finally AND operation is performed on both the results.

Now let us see the functions of these logical operators, that is, AND, OR and NOT.

1. **AND operator (&&)** – The AND operator returns true value when both the operands are true. The truth table for AND operator is given below :

Table 3.8 : Logical AND Operators

operand1	operand2	operand1 && operand2
True	True	True
True	False	False
False	True	False
False	False	False

2. **OR Operator (||)** – The OR operator (||) produces true output when one of the input is true or when both the inputs are true. The truth table of OR (||) operator is given below :

Table 3.9 : Logical OR Operators

operand1	operand2	operand1 || operand2
True	True	True
True	False	True
False	True	True
False	False	False

3. **NOT Operator (!)** – The NOT (!) operator is used to negate the condition, that is, if the true value is specified as an input then it produces false output and if false value is given as an input then true output is produced.

The truth table for NOT (!) operator is given below :

Table 3.10 : Logical XOR Operators

operand1	!operand
True	False
False	True

4. **Assignment Operators** – The assignment operator is used to assign a value to a variable. The syntax of assignment operator is given below :

varname= expr;

Where, varname is variable name and expr is an expression.

❑ **Check Your Progress – 4 :**

1. Write a note on Bitwise operators.

....

....

2. Unary operator perform operation based on ________ operand.

(A) One (B) two (C) three (D) four

3. The ________ operators are used to perform arithmetical operations. .

(A) Bitwise (B) Arithmetic (C) Relational (D) Logical

4. ________ also known as increment / decrement operator.

(A) ++ & – – (B) + & – (C) += & –= (D) =+ & =–

5. The ________ operator is used when the manipulation is to be done bit by bit.

(A) Bitwise (B) Arithmetic (C) Relational (D) Logical

6. The ________ operator is also known as ternary operator.

(A) Bitwise (B) conditional (C) Relational (D) Logical

7. The ________ have states and behaviours.

(A) Class (B) Objects (C) Variable (D) Method

8. Assignment operators are used to assign a value to operand.

(A) True (B) False

9. Bitwise Not operator is denote by ~ sign.

(A) True (B) False

10. The << operator (left shift) causes the bits of the left operand to be shifted to the left based on the value of the right operand.

(A) True (B) False

11. Logical operators are used to combine more than one condition to perform logical operation.

(A) True (B) False

3.8 Ternary Operator :

The conditional operator is also known as ternary operator. It is denoted by ? and :. The syntax of same can be given as :

```
expr1 ? expr2  :expr3
```

In the above syntax, expr1, expr2 and expr3 are expressions and expr1 must be of boolean type.

For example,

```
If a = 10
   b = 2
   z = (a > b) ? a : b
```

In the above example, the value of a > b gets evaluated first, if the condition is true then value of a gets assigned to z otherwise the value of b.

```
class TarnaryDemo
{
   public static void main(String args[])
   {
      int a = 12;
      int b = 8;
      int c;

      c =  (a > b) ? a : b;

      System.out.println(c);
   }
}
```

OUTPUT

```
12
```

3.9 Operator Precedence :

The precedence of operators is useful when there are several operators in an expression. Java has specific rules for determining the order of evaluation of an expression. The given table displays the list of operators in the order of precedence. The hierarchy of Java operators with highest precedence is shown first.

a. All those expressions which are inside parenthesis are first evaluated, the nested parenthesis are evaluated from the innermost parenthesis to the outer.

b. All the operators which are in the same row have equal precedence.

c. The given table shows the list of operators with their order of evaluation–

Table 3.11 : Operators and their Evaluation

Operator	Type	Order of Evaluation
( ) [ ] .	Parenthesis Array Subscript Member Access	Left to right
++, – –	Prefix increment, decrement	Right to left
++, – – –	Postfix Increment, decrement Unary minus	Right to left
*,/,%	Multiplicative	Left to right
+, –	Additive	Left to right
<, >, <=, >=	Relational	Left to right
==,!=	Equality	Left to right
&&	AND	Left to right
||	OR	Left to right
? :	Conditional	Right to left
=, +=, –+, *=,/+,%=	Assignment	Right to left

❖ **Java Programs :**

When we consider a Java program it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what do class, object, methods and instant variables mean.

- **Object** – Objects have states and behaviors. For example : A dog has states–color, name, and breed as well as behaviors –wagging, barking and eating. An object is an instance of a class.
- **Class** – A class can be defined as a template/ blue print that describe the behaviors/states that object of its type support.
- **Methods** – A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instant Variables** – Each object has its unique set of instant variables. An object's state is created by the values assigned to these instant variables.
- **Writing and Compiling Programs** – Let us look at a simple code that would print the word Welcome.

```
public class Welcome
   {
      /*This program will print Welcome */
      public static void main (String args [ ])
      {
         System.out.println ("Welcome"); //Print Welcome
      }
   }
```

Let us look at how to save the file, compile and run the program. Please follow the steps given below :

1. Open notepad and add the code as above.
2. Save the file as : Welcome.java.
3. Open a command prompt window and go to the directory where you saved the class. Assume its C:\.
4. Type ' javac Welcome.java ' and press enter to compile your code. If there are no syntax errors in your code the command prompt will take you to the next line (Assumption : The path variable is set).
5. Now type ' java Welcome ' to run your program.
6. You will be able to see Welcome' printed on the window. C:> javac Welcome.java

   C:>java Welcome Welcome

- **Basic Syntax –** About Java programs, it is very important to keep in mind the following points.
- **Case Sensitivity –** Java is case sensitive which means identifier Hi and hi would have different meaning in Java.
- **Class Names –** For all class names the first letter should be in Upper Case.

  If several words are used to form a name of the class, each inner word's first letter should be in upper case.

  Example : class Welcome
- **Method Names –** All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

  Example : public void myMethodName()
- **Program File Name –** Name of the program file should exactly match the class name.

When saving the file you should save it using the class name (Remember java is case sensitive) and append '.java' to the end of the name. (If the file name and the class name do not match your program will not compile).

Example : Assume 'Welcome' is the class name. Then the file should be saved as 'Welcome.java'

- **Public static void main(String args[ ]) –** Processing of java program starts from the main() method which is a mandatory part of every java program.

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are :

1. Visible to the package, its default modifier.
2. Visible to the class only (private)
3. Visible to the world (public)
4. Visible to the package and all subclasses (protected).

Default Access Modifier – No keyword – Default access modifier means we do not explicitly declare an access modifier for a class, field, method etc.

A variable or method declared without any access control modifier is available to any class in the same package. The default modifier cannot be used for methods, fields in an interface.

Example :

Variables and methods can be declared without any modifiers, as in the following example :

```
String x= "123";
boolean processorder ( )
{
return true;
}
```

Private Access Modifier – private – Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself.

Private access modifier is the most restrictive access level. Class and interfaces cannot be private.

Variables that are declared private can be accessed outside the class if public getter methods are present in the class.

Using the private access modifier is the main way that an object encapsulates itself and hides data from the outside world.

So to make the variable available to the outside world, we defined two public methods : get Format(), which returns the value of format and set Format(String), which sets its value.

Public Access Modifier – public – A class, method, constructor, interface etc declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.

However, if the public class we are trying to access is in a different package then the public class still need to be imported.

Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.

Example :

The following function uses public access control :

```
Public static void main (String args [ ])
{
//………..
}
```

The main ( ) method of an application has to be public. Otherwise, it could not be called by a Java interpreter (such as java) to run the class.

Protected Access Modifier – protected – Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.

The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected; however, methods and fields in an interface cannot be declared protected.

Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

3.10 Let Us Sum Up :

An operator is used to perform specific operation on two or more operands. The operators are classified as given are Arithmetic Operators, Relational Operators, Logical Operators, Assignment Operators, Increment and Decrement Operators, Conditional Operators, Bitwise Operators, Special Operators

The precedence of operators is useful when there are several operators in an expression. Java has specific rules for determining the order of evaluation of an expression. The given table displays the list of operators in the order of precedence. The hierarchy of Java operators with highest precedence is shown first as all those expressions which are inside parenthesis are first evaluated, the nested parenthesis are evaluated from the innermost parenthesis to the outer. Secondly all the operators which are in the same row have equal precedence.

Let us talk about our understanding related to Java program it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what do class, object, methods and instant variables mean.

- **Object** – Objects have states and behaviors. For example : A dog has states–color, name, breed as well as behaviors –wagging, barking and eating. An object is an instance of a class.
- **Class** – A class can be defined as a template/ blue print that describe the behaviors/states that object of its type support.
- **Methods** – A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instant Variables** – Each object has its unique set of instant variables. An object's state is created by the values assigned to these instant variables

3.11 Suggested Answer for Check Your Progress :

❑ **Check Your Progress 1 :**

See Section 3.2

❑ **Check Your Progress 2 :**

See Section 3.3

❑ **Check Your Progress 3 :**

See Section 3.5

❑ **Check Your Progress 4 :**

1 : See Section 3.7		**2 :** A	**3 :** B	**4 :** A
5 : A	**6 :** B	**7 :** B	**8 :** A	**9 :** A
10 : A	**11 :** A			

3.12 Glossary :

1. **Operators** – java provides six kind of operators. Arithmetic's operatos uses for the mathematic operation. It required two operands to perform operation. Increment and Decrement are the unary operators. It will either increase or decrease operand value by one. Assignment operator is responsible to assign value of right hand operand to left hand side operand. Bitwise operators work on bit. It basically use for binary operation. Relation operator is uses for the compression between two operands. Logical operator's uses to take decision based on operands values. Ternary operators required three operands and uses for the decision making.
2. **Object** – Objects have states and behaviors. For example : A dog has states–color, name, breed as well as behaviors –wagging, barking and eating. An object is an instance of a class.
3. **Class** – A class can be defined as a template/ blue print that describe the behaviors/states that object of its type support.
4. **Methods** – A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.

3.13 Assignment :

Write a program to explain the use of operators.

3.14 Activities :

1. Explain the concept of Pre-Increment/Decrement and Post Increment/ Decrement.
2. Explain class, object, methods and instant variables

3.15 Case Study :

For our case study, we will be creating two classes. They are Student and Student Details.

First open notepad and add the following code. Remember this is the Student class and the class is a public class. Now, save this source file with the name Student.java.

The Student class has four instance variables name, age, course and semester.

The class has one parameterised constructor, which takes parameters.

The class should also comprise with one public method display(), which displays the details of the student.

Processing starts from the main method. Therefore in–order for us to run this Student class there should be main method and objects should be created. We will be creating a separate class for these tasks.

Student Details class, which creates two instances of the class Student and invokes the methods for each object to assign values for each variable.

3.16 Further Readings :

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000
2. Java 2, The Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999
3. Programming with Java, Ed. 2, E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000
4. The Java Tutorial, Ed. 2, 2 volumes, Mary Campione and Kathy Walrath, Addison Wesley Longmans, 1998
5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele & Gilad Bracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000
6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition, 2000

Unit 04

LOOPS AND SELECTION STATEMENTS

UNIT STRUCTURE

4.0 Learning Objectives :

After learning this unit, you will be able to understand :

- Explain loops – for, while and do–while loop
- Describe nested loops
- Discuss Selection statement – if, if else, nested if, switch statement and recursion
- Define Arrays – one dimensional and multidimensional
- Illustrate switch statement

4.1 Introduction :

A programming loop is a control structure that allows a programmer to execute the same instruction or group of instructions over and over until some condition is met. All loops have a basic structure to them, though there are many types.

We can use Java JDBC Select statement in a java program to retrieve the data and display it for the respective Tables. JDBC returns results in a Result Set object, so we need to declare an instance of the class Result Set to hold our results. Select is the SQL keyword that performs a query

4.2 Loops :

The process of executing a block of statements a number of times is known as looping or iterating. There are mainly three types of loops in Java which are used in programs when the same set of statements are executed a number of times.

❖ For Loop :

The for loop is used to execute the same set of statements a number of times. With for loop, the initial value of variable is specified with the condition which gets checked so that the given set of statements be executed till the value of variable is less than/greater than or equals to it, the increment/ decrement value of the variable is also specified which keeps on increasing/ decreasing the value of variable on each iteration.

The syntax of for loop is given below :

```
for (initial value; condition; increment/decrement value);
{ // start of for loop
//statements to be executed
}//end of for loop
```

The flow chart for execution of for loop is given below :

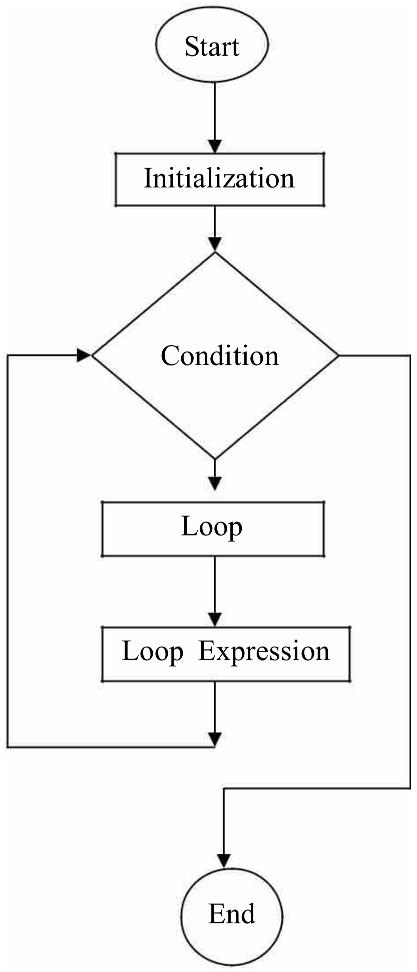


Figure 4.1 Flow chart of For loop

False

True

For example,

```
for ( int i=1; i<=5; i++)
{
System.out.println ( i );
}
```

The above code fragment will display numbers from 1 to 5.

Here is the flow of control of For Loop –

1. The initialization step is executed first and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
2. Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement past the for loop.
3. After the body of the for loop executes, the flow of control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the Boolean expression.
4. The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the for loop terminates.

❖ **While Loop**

The while loop is used to execute the same set of statements a number of times. In while loop, first the condition is specified with while and the statements specified under it gets executed a number of times. The syntax of same is given below :

```
while (condition)
{
//loop statements
//increment/decrement value
}//end of while loop
```

The execution of statements in while loop is illustrated using the given flowchart :

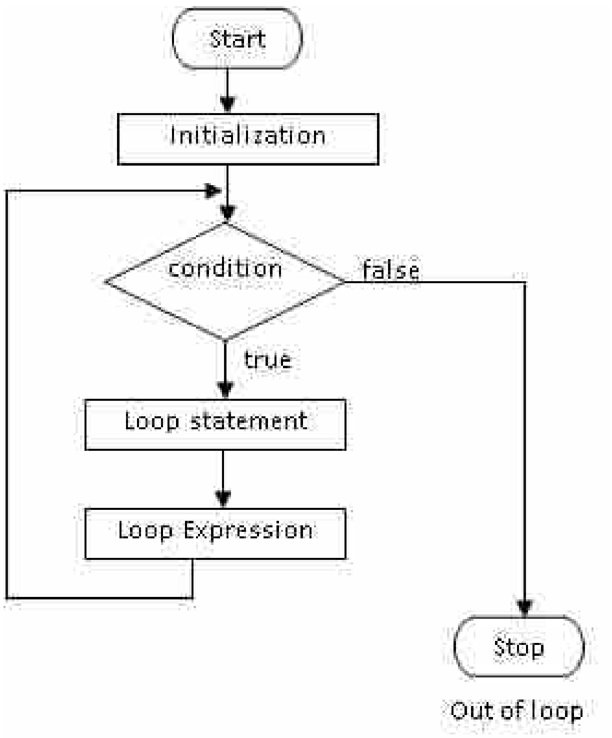


Figure 4.2 Flowchart for while loop

The statements within the while loop gets executed till the condition being tested remains true. As soon as it becomes false, the control of the program passes to the first statement that follows the body of the loop.

If the statements within the parenthesis of while loop is single then the use of parenthesis is optional.

❖ **Do – While Loop**

Just like while loop, the do–while loop is also used to execute the same set of statements a number of times. The syntax of do–while is given below :

```
Initialisation
do
{
//increment / Decrement Value ;
}while(condition)
value; } while (condition);
```

The process of execution of statements of do–while loop is illustrated below :

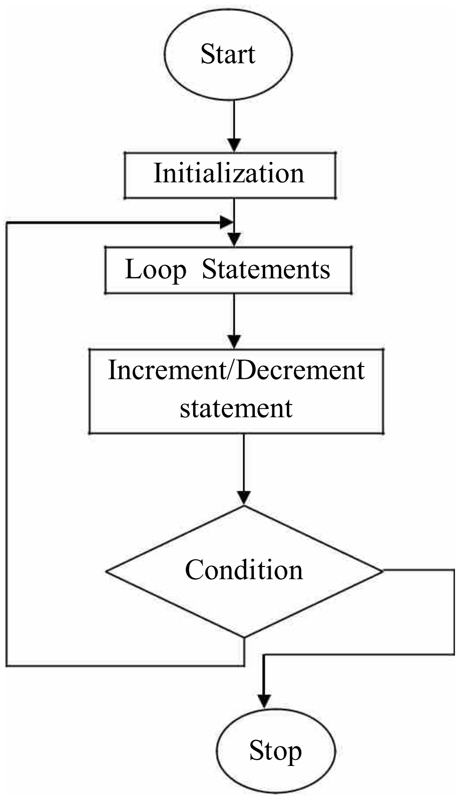


Figure 4.3 Flow chart for Do while loop

The while loop is also called entry controlled loop whereas the do–while loop is called as exit–controlled loop. There is a small difference between both these loops. The while loop first tests the condition and then executes the statements and do–while loop first executes the statements and then checks the condition. The differences between them are shown below in the given table :

Table 4.1 : Do–while loop Vs While loop

Do-while loop	While loop
Also called exit–controlled loop	Also called entry controlled loop
In do–while loop, first the statements get executed and then the condition is checked	First the condition gets checked and then the statements are executed
The statements get executed even if the condition is wrong	The statements do not get executed if the condition is wrong

❑ **Check Your Progress – 1 :**

1. Give the difference between do while and while loop.
2. Explain for loop with the help of flowchart.

....

....

....

....

....

4.3 Nested Loops :

A loop within a loop is called nested loop. When two loops are nested, the outer loop takes control of the number of complete repetitions of the inner loop. While all types of loops may be nested, the most commonly used nested loop is for loop.

Following Example show the example of Nested Loops :

Example : /* Program print the following output

```
1
1 2
1 2 3
1 2 3 4
*/

class Demoof_nestedloop
{
 public static void main(String args[])
{
 for (int row = 1 ; row <= 4 ; row++)
{
 for(int col = 1 ; col <= row ; col++)
 {
   System.out.print(col + "\t");
  }
  System.out.println(" ");
 }
}
}
```

Output

```
1
1   2
1   2   3
1   2   3   4
```

When working with nested loop, the outer loop is only changed once the inner loop is completely finished.

❑ **Check Your Progress – 2 :**

1. Explain nested loops.
2. Which is the commonly used nested loop ?

....

....

....

....

....

4.4 Selection Statements :

The selection statements are used for decision making purpose. These statements are discussed below :

❖ **If Statement**

The 'if' statement is used to check a condition; if that condition is true then the statement specified after if statement gets executed. The syntax of if statement is given below :

```
if (condition)
{
//statement 1
//statemen
……..
……..
//statement n
}
```

If there is a single statement then we need not to specify the braces but if there is more than one statement then it has to be enclosed in pair of braces. The process flow of if statement is explained in the given flowchart :

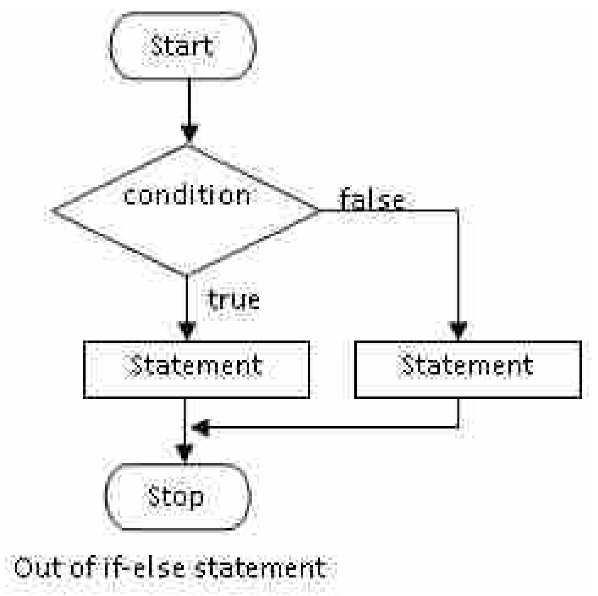


Figure 4.4 out of it statement

❖ **The If – Else Statement**

The if–else statement is also used to check a condition just like if statement but if the given statement is not true then the statements specified in the else part gets executed.

The syntax of if–else statement is given below :

```
if ( condition)
{
//statements
//statements
}

else

{
//statements
//statements
}
}
```

❖ **Nested If**

An if inside another if is called as nested if statement. The syntax of nested if statement is given below :

```
if (condition)
{
 if (condition)
 {
  //statements
 }
 //statements
}
else
{
 if (condition)
 {
  //statements
 }
 //statements
}
```

❖ **Switch Statement**

The switch statement tests the value of a variable and based on that executes the corresponding case statement. The last statement of switch–case statement contains a default statement, which gets executed when all the case statements specified does not match with the value of the variable specified.

The syntax of switch–case statement is given below :

```
switch (expression)
{
case cond1:
 code_block_1;
case cond2:
 code_block_2;
...
...
case condn:
 code_block_n;
default:
 code_block_default;
}
```

Here, Here, In above syntax switch(expression), where expression is a variable either of integer, character, short etc. type of data type. In each case statement within the switch statement, a comparison is made with the value entered by the user and the value specified with case. If the comparison evaluates to true, the code specified within that block is executed, otherwise it goes to next case statement. The last default statement gets executed if none of the conditions match.

The following rules apply to a switch statement :

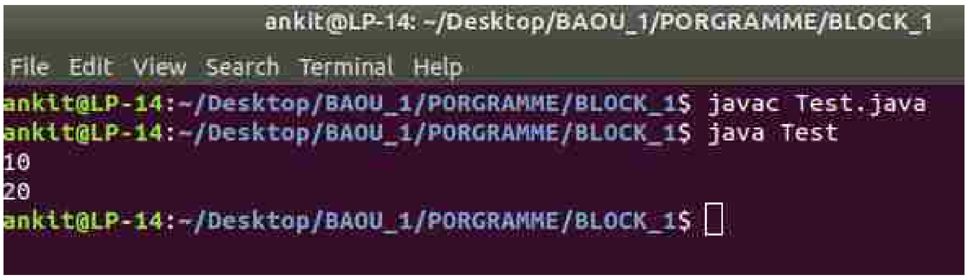
- The variable used in a switch statement can only be a byte, short, int, or char.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared and followed by a colon symbol.
- The value for a case must be the same data type as the variable in the switch and it must be a constant or a literal.

- When any switch case is true then, the statements following that case will execute until a break statement is reached.
- When a break statement is reached, the switch terminates and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.
- A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

❖ The Break Statement

The break statement sends the control of program out of the loop. This statement is useful when in certain instances we want to exit out of the loop instantly. As soon as this keyword is encountered inside any loop, the control of the program automatically passes to the next statement after the loop.

```
public class Test
{
 public static void main(String args[])
 {
  int[] numbers = {10,20,30,40,50};

  for(int x : numbers)
  {
   if( x == 30)
   {
    break;
   }

   System.out.print(x);
   System.out.print("\n");
  }
 }
}

//OUTPUT
10
20
```


ankit@LP-14: ~/Desktop/BAOU_1/PORGRAMME/BLOCK_1
File Edit View Search Terminal Help

```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_1$ javac Test.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_1$ java Test
10
20
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_1$
```

Figure 4.5 Output of Program

❖ **The Continue Statement**

The continue statement sends the control of program to the beginning of the loop. As soon as this statement occurs in the program, the rest of the statements written after continue statement is bypassed and the control of the program is sent to the beginning of the loop.

For example,

```
//Code to print numbers from 1 to 5 except 4
int i=1
while (i<=5)
{
   if (i==4)
      continue;
   System.out.println (i);
   i++;
}
```

❖ **Recursion**

Recursion is kind of process that Java function call itself. The function body contain the statement that call itself. The function body must have conditional statement that will stop calling of function when conditional statement become true. This might be true is some cases but in practice, we can check to see if a certain condition is true and in that case exit (return from) our method. The case in which we end our recursion is called a base case. Additionally, just as in a loop, we must change some value and incremently advance closer to our base case.

Consider this method :

```
Void myMethod (int    counter)
{
      if(counter == 0)
            return;
         else{
         System.out.println("hello" + counter);
         myMethod(—counter);
         System.out.println(""+counter);
         return;
         }
}
```

If the method is called with the value 4, what will the output be ? Explain. The above recursion is essentially a loop like a for loop or a while loop. When do we prefer recursion to an iterative loop ? We use recursion when we can see that our problem can be reduced to a simpler problem that can be solved after further reduction.

Every recursion should have the following characteristics :

1. A simple base case which we have a solution for and a return value.
2. A way of getting our problem closer to the base case, i.e., a way to chop out part of the problem to get a somewhat simpler problem.
3. A recursive call which passes the simpler problem back into the method.

The key to thinking recursively to see the solution to the problem as a smaller version of the same problem. The key to solving recursive programming requirements is to imagine that your method does what its name says it does even before you have actually finish writing it. You must pretend the method does its job and then use it to solve the more complex cases. The same is explained below :

Identify the base case(s) and what the base case(s) do. A base case is the simplest possible problem (or case) your method could be passed. Return the correct value for the base case. Your recursive method will then be comprised of an if–else statement where the base case returns one value and the non–base case(s) recursively call(s) the same method with a smaller parameter or set of data. Thus, you decompose your problem into two parts : (1) The simplest possible case which you can answer (and return for) and (2) all other more complex cases which you will solve by returning the result of a second calling of your method.

This second calling of your method (recursion) will pass on the complex problem but reduced by one increment. This decomposition of the problem will actually be a complete, accurate solution for the problem for all cases other than the base case. Thus, the code of the method actually has the solution on the first recursion.

❑ **Check Your Progress – 3 :**

1. Explain the characteristics of recursion.
2. Write the rules for switch statement.

....

....

....

....

....

4.5 Arrays :

An array is a collection of similar types of variables which are referenced under a single name. It can also be defined as a collection of homogeneous cells inside computer's memory.

To understand the concept, let us take an example. Consider that you want to store the marks of 50 students and display them. Now, the simple way which you have studied till now is to take 50 separate variables, store

50 numbers in them and then display their values. But this will make your program very lengthy and complicated, so, in order to reduce the number of statements in a program, the concept of arrays is used.

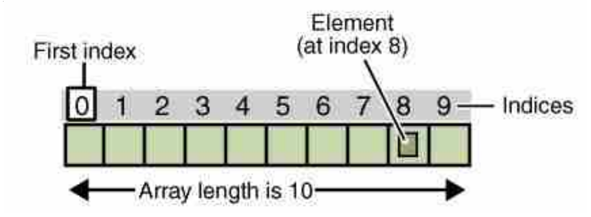


Figure 4.6 Array index

Each item of an array is called an element and each element is accessed by its index number. As shown in the above figure, the numbering begins with 0 and the total length of the array is 10. So, the 1st element is stored at 0th index, 2nd at 1st index and so on. The last element that is 10th element is stored at 9th index.

These arrays are classified as :

a. One–dimensional array

b. Multi–dimensional array

a. **One–Dimensional Arrays –** A one–dimensional array contains single row or column. The syntax of declaring single–dimensional array is :

data type varname[size];

Here, type declares the data type of the array, varname is the name of the array and size is the total size of the array.

An array can also be declared as :

int student[]= new int [5];

int [ ] student = new int [5];

Processing Arrays – When processing array elements, we often use either for loop or foreach loop because all of the elements in an array are of the same type and the size of the array is known.

Example :

Here, is a complete example of showing how to create, initialise and process arrays :

```
public class testarray
{
public static void main (String args[ ])
{
double mylist [ ] = { 4, 6,2,9,7 };
//Print all the array elements
for (int i=0;i< myList.length;i++)
{
System.out.println (myList[i] + " ");
}
//Summing all elements
double total=0;
for( int i=0; i<myList.length;i++)
{
total+= myList[i];
}
System.out.println ("Total is" + total);

//Finding the largest element
double max=myList[0];
for (int i=1; i<myList.length;i++)
{
if (myList[i] > max)
max = myList[i];
}
System.out.println ("Max is" + max);
}
}
```

This would produce following result :

```
4
6
2
9
7
Total is 28
Max is 9
```

Passing Arrays to Methods – Just as you can pass primitive type values to methods, you can also pass arrays to methods. For example, the following method displays the elements in an int array :

```
public static void printArray (int [ ] array)
{
for (int i=0; i<array.length;i++)
{
System.out.println (array[i] + " " );
}
}
```

You can invoke it by passing an array. For example, the following statement invokes the printArray method to display 3, 1, 2, 6, 4 and 2 :

```
printArray (new int [ ] { 3, 2, 5, 6,9} );
```

The Arrays Class – The java. util. Arrays class contains various static methods for sorting and searching arrays, comparing arrays and filling array elements. These methods are overloaded for all primitive types.

SN	Methods with Description
1.	public static int binarySearch(Object[] a, Object key) Searches the specified array of Object (Byte, Int , double etc) for the specified value using the binary search algorithm. The array must be sorted prior to making this call. This returns index of the search key, if it is contained in the list; otherwise, (–(insertion point + 1).
2.	public static boolean equals(long[] a, long[] a2) Returns true if the two specified arrays of longs are equal to one another. Two arrays are considered equal if both arrays contain the same number of elements and all corresponding pairs of elements in the two arrays are equal. This returns true if the two arrays are equal. Same method could be used by all other primitive data types (Byte, short, Int etc.)
3.	public static void fill(int[] a, int val) Assigns the specified int value to each element of the specified array of ints. Same method could be used by all other primitive data types (Byte, short, Int etc.)
4.	public static void sort(Object[] a) Sorts the specified array of objects into ascending order, according to the natural ordering of its elements. Same method could be used by all other primitive data types (Byte, short, Int etc.)

b. **Multi–dimensional Arrays –** If an array contains multiple rows and multiple columns then it is called as multi–dimensional array or two–dimensional array.

The syntax of declaring two–dimensional array is :

data type array name[][] = new int [size of row][size of column];

The total number of elements which can be stored in 2–D array can be obtained by the given formula :

Size of the array=number of rows X number of columns

For example,

int student[][] = new int [3][4]

The above statement creates a two–dimensional array of student name of integer data type with 3 rows and 4 columns, that is, the total number of elements which can be stored in this array is 12.

❑ Check Your Progress – 4 :

1. Explain the process of passing arrays to methods.
2. Discuss the process of using methods in arrays.

....

....

....

....

....

3. The process of executing a block of statements a number of times is known as ________.

   (A) Method (B) Class

   (C) Loop (D) Decision Making

4. ________ loop is called as exit control loop.

   (A) for (B) if (C) while (D) do while

5. Loop within loop is called ________ loop.

   (A) Nested (B) within (C) for (D) simple

6. The selection statements are used for ________ purpose.

   (A) Looping (B) Coding

   (C) Decision making (D) processing

7. if the given statement is not true then the statements specified in the ________ part gets executed.

   (A) for (B) while (C) if (D) else

8. The ________ statement sends the control of program out of the loop

   (A) Continue (B) break (C) if (D) else

9. If the statements within the parenthesis of while loop is single then the use of parenthesis is optional.

   (A) True (B) False

10. While loop is entry control loop.

    (A) True (B) False

11. When working with nested loop, the outer loop is only changed once the inner loop is completely finished.

    (A) True (B) False

12. Each 'if' statement must have 'else' statement.

    (A) True (B) False

4.6 Let Us Sum Up :

This gave us lot of insight in to the Java programming aspects with clarity such as loop, the process of executing a block of statements a number of times is known as looping or iterating. There are mainly three types of loops in Java which are used in programs when the same set of statements are executed a number of times.

1. **FOR LOOP –** The for loop is used to execute the same set of statements a number of times. With for loop, the initial value of variable is specified with the condition which gets checked so that the given set of statements be executed till the value of variable is less than/greater than or equals to it, the increment/decrement value of the variable is also specified which keeps on increasing/decreasing the value of variable on each iteration. We need to understand and apply control of FOR LOOP.
2. **WHILE LOOP –** The while loop is used to execute the same set of statements a number of times. In while loop, first the condition is specified with while and the statements specified under it gets executed a number of times.
3. **DO–WHILE LOOP –** Just like while loop, the do–while loop is also used to execute the same set of statements a number of times. The syntax of do–while is given below :

4. **NESTED LOOP –** A loop within a loop is called nested loop. When two loops are nested, the outer loop takes control of the number of complete repetitions of the inner loop. While all types of loops may be nested, the most commonly used nested loop is for loop. When working with nested loop, the outer loop is only changed once the inner loop is completely finished.
5. **THE IF–ELSE STATEMENT –** The if–else statement is also used to check a condition just like if statement but if the given statement is not true then the statements specified in the else part gets executed.
6. **NESTED IF –** If inside another if is called as nested if statement.
7. **SWITCH STATEMENT –** The switch statement tests the value of a variable and based on that executes the corresponding case statement. The last statement of switch–case statement contains a default statement, which gets executed when all the case statements specified does not match with the value of the variable specified. At this point of time we need to understand and follow rules to apply to a switch statement :
8. **THE BREAK STATEMENT –** The break statement sends the control of program out of the loop. This statement is useful when in certain instances we want to exit out of the loop instantly. As soon as this keyword is encountered inside any loop, the control of the program automatically passes to the next statement after the loop.
9. **THE CONTINUE STATEMENT –** The continue statement sends the control of program to the beginning of the loop. As soon as this statement occurs in the program, the rest of the statements written after continue statement is bypassed and the control of the program is sent to the beginning of the loop.
10. **RECURSION –** Recursion is when a function calls itself. That is, in the course of the function definition there is a call to that very same function. At first this may seem like a never ending loop, or like a dog chasing its tail. It can never catch it. So too it seems our method will never finish. This might be true is some cases but in practice, we can check to see if a certain condition. is true and in that case exit (return from) our method. The case in which we end our recursion is called a base case. Additionally, just as in a loop, we must change some value and incrementally advance closer to our base case.
11. **ARRAY –** An array is a collection of similar types of variables which are referenced under a single name. It can also be defined as a collection of homogeneous cells inside computer's memory. a. One–dimensional array and b. Multi–dimensional array
12. **PROCESSING ARRAYS –** When processing array elements, we often use either for loop or for each loop because all of the elements in an array are of the same type and the size of the array is known.
13. **PASSING ARRAYS TO METHODS –** Just as you can pass primitive type values to methods, you can also pass arrays to methods. For example, the following method displays the elements in an int array :
14. **THE ARRAYS CLASS –** The java. util. Arrays class contains various static methods for sorting and searching arrays, comparing arrays and filling array elements. These methods are overloaded for all primitive types.

15. **MULTI–DIMENSIONAL ARRAYS** – If an array contains multiple rows and multiple columns then it is called as multi–dimensional array or two–dimensional array.

4.7 Suggested Answer for Check Your Progress :

❑ **Check Your Progress 1 :**

See Section 4.2

❑ **Check Your Progress 2 :**

See Section 4.3

❑ **Check Your Progress 3 :**

See Section 4.4

❑ **Check Your Progress 4 :**

See Section 4.5

❑ **Check Your Progress 5 :**

1 : See Section 4.6		**2 :** A	**3 :** D	**4 :** A
5 : C	**6 :** D	**7 :** B	**8 :** A	**9 :** A
10 : A	**11 :** B			

4.8 Glossary :

1. **For Loop** – The For loop is used to execute the same set of statements a number of times.
2. **While Loop** – The while loop is used to execute the same set of statements a number of times.
3. **Do–while loop** – Just like while loop, the do–while loop is also used to execute the same set of statements a number of times.
4. **Nested loop** – A loop within a loop is called nested loop. When two loops are nested, the outer loop takes control of the number of complete repetitions of the inner loop.
5. **The if–else Statement** – The if–else statement is also used to check a condition just like if statement but if the given statement is not true then the statements specified in the else part gets executed.
6. **Nested if** – if inside another if is called as nested if statement.
7. **Switch Statement** – The switch statement tests the value of a variable and based on that executes the corresponding case statement
8. **The Break Statement** – The break statement sends the control of program out of the loop.
9. **The Continue Statement** – The continue statement sends the control of program to the beginning of the loop.
10. **Recursion** – Recursion is when a function calls itself. That is, in the course of the function definition there is a call to that very same function.
11. **Array** – An array is a collection of similar types of variables which are referenced under a single name. It can also be defined as a collection of homogeneous cells inside computer's memory. a. One–dimensional array and b. Multi–dimensional array

12. **Processing Arrays –** When processing array elements, we often use either for loop or for each loop because all of the elements in an array are of the same type and the size of the array is known.
13. **Passing Arrays to Methods –** Just as you can pass primitive type values to methods, you can also pass arrays to methods. For example, the following method displays the elements in an int array :
14. **The Arrays Class –** The java.util.Arrays class contains various static methods for sorting and searching arrays, comparing arrays and filling array elements. These methods are overloaded for all primitive types.
15. **Multi–dimensional Arrays –** If an array contains multiple rows and multiple columns then it is called as multi–dimensional array or two–dimensional array.

4.9 Assignment :

1. Explain the flow of control of for loop.
2. Explain the flow of control of while loop.
3. Explain the flow of control of do while loop
4. Explain the flow of if condition

4.10 Activities :

1. Using switch case statement, write a program for calculator.
2. Write a java program that will print "BAOU" ten time on screen.
3. Write a java program that will store the age of ten person on array. Read the age one by one and if the age is less than 18 years then print message "you are child" otherwise print "you are young person".
4. Write a java program that will store 10 english characters in array. Read it one by one if the character is any vowel then print message "The character is vowel" otherwise print the character only.
5. Write a java program that will store 5 float values in array. Display the average of the value.

4.11 Case Study :

What are selection statements explain with the help of an example.

4.12 Further Reading :

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000
2. Java 2, The Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999
3. Programming with Java, Ed. 2,E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000
4. The Java Tutorial, Ed. 2, 2 volumes, Mary Campione and Kathy Walrath, Addison Wesley Longmans, 1998
5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele & Gilad Bracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000
6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition, 2000

BLOCK SUMMARY :

This Unit No. 1 gives lot of basic learning right from History of Java that James Gosling initiated the Java language project in June 1991 for use in one of his many set–top box projects. The language, initially called Oak after an oak tree that stood outside Gosling's office, also went by the name Green and ended up later renamed as Java. As the current programming problems are complex, as beyond a certain size, structured programming cannot manage complexity. Using the concept of object oriented programming, complex programs can be organized using classes, inheritance and polymorphism. C++ was one of the popular

About Garbage collection in C++ language, the dynamically allocated objects are manually released by the use of a delete operator. In Java, the deallocation process is automatically done and the technique to do that is called garbage collection.

Creating a Java Program is the execution of Java Program is divided into several steps.

The Unit No. 2 deals with several important basic aspects of Java Programming. One of them is the tokens of a language which are the basic building blocks and can be put together to construct programs. These tokens are explained below :

Identifiers – Identifiers are those words, which help to identify an entity. It can be used for class names, method names and variable names. It can be represented using uppercase and lowercase characters, numbers, underscore and dollar sign characters :

Java is a programming language used to create programs that can run on a variety of Operating Systems. A Java constant is a variable with a pre–defined value. Although Java does not have a constant type, you can effectively obtain the same effect with a final variable. This enables you to have a control of what is constant and what is not. We learned that there is Standard Naming Convention. In declaring Java constant variables, you should declare the variable names in ALL CAPS (notice each letter of the variable name above). The words in Java constants are typically separated with underscores (as in the example above). This format indicates that these values are constants. It will be easier for an individual to read a code if this standard naming convention is followed.

An operator is used to perform specific operation on two or more operands. The operators are classified as given are Arithmetic Operators, Relational Operators, Logical Operators, Assignment Operators, Increment and Decrement Operators, Conditional Operators, Bitwise Operators, Special Operators

The precedence of operators is useful when there are several operators in an expression.

Java Programs – When we consider a Java program it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what do class, object, methods and instant variables mean. Apart from this we have understood

Tokens, Identifiers, Literals/Constants, Typecasting, Typecasting, Object, Class, Methods

Unit no. 3, this gave us lot of insight in to the Java programming aspects with clarity such as loop, the process of executing a block of statements a number of times is known as looping or iterating. There are mainly three types of loops in Java which are used in programs when the same set of statements are executed a number of times.

For loop – The For loop is used to execute the same set of statements a number of times.

Here is the flow of control of For loop : 1) The initialisation step is executed first and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears. 2) Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement past the for loop.3) after the body of the for loop executes, the flow of control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the Boolean expression.

The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the for loop terminates.

While loop – The while loop is used to execute the same set of statements a number of times. Then we understood Do–while loop – Just like while loop, the do–while loop is also used to execute the same set of statements a number of times. Next loop is Nested loop – A loop within a loop is called nested loop. When two loops are nested, the outer loop takes control of the number of complete repetitions of the inner loop. Then comes The if–else statement – The if–else statement is also used to check a condition just like if statement but if the given statement is not true then the statements specified in the else part gets executed.

Nested if – if inside another if is called as nested if statement. Switch Statement – The switch statement tests the value of a variable and based on that executes the corresponding case statement. Next loop is The Break Statement – The break statement sends the control of program out of the loop. The Continue Statement – The continue statement sends the control of program to the beginning of the loop. Recursion – Recursion is when a function calls itself. Array – An array is a collection of similar types of variables which are referenced under a single name. It can also be defined as a collection of homogeneous cells inside computer's memory. a. One–dimensional array and b. Multi–dimensional array. Then comes Processing Arrays – When processing array elements, we often use either for loop or for each loop because all of the elements in an array are of the same type and the size of the array is known. Then there is a loop called Passing Arrays to Methods – Just as you can pass primitive type values to methods, you can also pass arrays to methods. For example, the following method displays the elements in an int array : lastly we understood The Arrays Class – The java.util.Arrays class contains various static methods for sorting and searching arrays, comparing arrays and filling array elements. These methods are overloaded for all primitive types. Finally we have learned Multi–dimensional Arrays – If an array contains multiple rows and multiple columns then it is called as multi–dimensional array or two–dimensional array.

BLOCK ASSIGNMENT :

❖ **Short Questions :**

1. Write History of Java
2. Write the Importance of Java programming
3. What is meant by Garbage collection
4. Write the procedure for Java program
5. What do you mean by token
6. What do you mean by Identifier

❖ **Long Questions :**

1. Write a note on various operators
2. Write notes on Tokens, Identifiers, Literals/Constants, Typecasting, Typecasting, Object, Class, Methods
3. Write a note on various types of loops with its relevant examples.
4. What do you mean by class, object, methods and instant variables
5. What do you mean by loop statement

BIBLIOGRAPHY

http://www.tutorialspoint.com/java/java_overview.htm

http://en.wikipedia.org/wiki/Java_(programming_language)

http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

http://www.sunrays.co.in/Home/applied–core–java/tutorial/java–platform

http://cs.fit.edu/~ryan/java/language/basics.html

http://en.wikipedia.org/wiki/Coding_conventions

http://en.wikibooks.org/wiki/Introduction_to_Software_Engineering/Implementation/Code_Convention

http://www.symatech.net/java–constant

http://stackoverflow.com/questions/215497/in–java–whats–the–difference–between–public–default–protected–and–private

http://www.jdbc–tutorial.coms/jdbc–tutorials/jdbc–select–statement–example

http://danzig.jct.ac.il/java_class/recursion.html

❖ **Enrolment No. :** ☐

1. How many hours did you need for studying the units ?

Unit No.	1	2	3	4
No. of Hrs.				

2. Please give your reactions to the following items based on your reading of the block :

Items	Excellent	Very Good	Good	Poor	Give specific example if any
Presentation Quality	☐	☐	☐	☐	
Language and Style	☐	☐	☐	☐	
Illustration used (Diagram, tables etc)	☐	☐	☐	☐	
Conceptual Clarity	☐	☐	☐	☐	
Check your progress Quest	☐	☐	☐	☐	
Feed back to CYP Question	☐	☐	☐	☐	

3. Any other Comments

....

....

....

....

....

....

....

....

Dr. Babasaheb Ambedkar Open University Ahmedabad

BCAR-204/ DCAR-204

Object Oriented Concepts & Programming–1 (Core Java)

BLOCK 2 : OBJECT, CLASSES AND FEATURES

OBJECT, CLASSES AND FEATURES

Block Introduction :

In this Block, we will be able to understand the use and constructs of objects, classes, passing arguments to methods, constructors, use of constructors, overloading constructors, why finalize method, keywords and how and why to use 'this' keyword. We also study the use of various methods available under the Wrapper Classes. The Collection Framework will give us an idea about how to implement the Data Structure using Java.

Block Objectives :

After learning this Block, you will be able to understand :

- Discuss the general form of class.
- Discuss the Wrapper Classes
- Discuss the Collection Framework Classes
- Describe argument passing.
- Define constructors.
- Explain the keyword and finalize () method.

Block Structure :

Unit 05

OBJECTS AND CLASSES

UNIT STRUCTURE

5.0 Learning Objectives :

After learning this unit, you will be able to :

- Discuss the general form of class.
- Describe argument passing.
- Define constructors.
- Explain the 'this' keyword and finalise ( ) method.

5.1 Introduction :

"A class is nothing but a blueprint or a template for creating different objects which defines its properties and behaviors. Java class objects exhibit the properties and behaviors defined by its class. A class can contain fields and methods to describe the behavior of an object."

"Methods are nothing but members of a class that provide a service for an object or perform some business logic. Java fields and member functions names are case sensitive. Current states of a class's corresponding object are stored in the object's instance variables. Methods define the operations that can be performed in java programming."

5.2 The General Form of a Class :

❖ **Object Oriented Concepts :**

An object is an identifiable entity with some characteristic features and behavior. Anything which has some properties and performs some behavior is called an object.

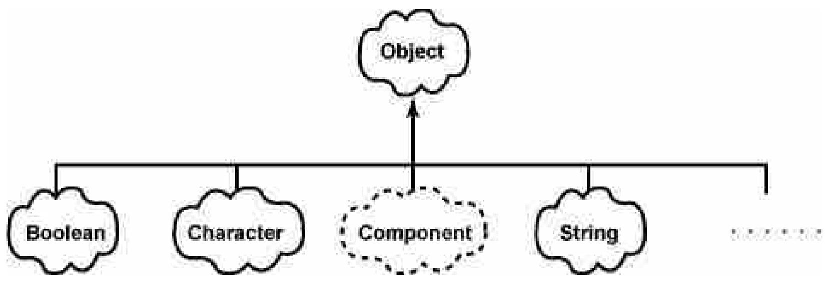


Figure 5.1 Object Oriented Concepts

For Example :

Pen is an object, as its characteristic features are its shape, colour etc. that is, it's cylindrical in shape and blue in colour and the characteristic behavior is the purpose of using it, that is, it is used for writing.

Consider another Example : Chair, Table, Eraser, Note Book all are objects, as all of them have some characteristic feature and perform some behavior.

❖ **Classes :**

Collection of similar types of objects is called Class. A Class is also called as an Object Factory as once the class is created we can create as many objects as we wish using that class.

The concept of Class will become clearer with the help of this Example : Consider that in a school, in drawing class, the teacher has a sample copy of a card (can be used for birthdays, anniversary etc.) which has to be made by each and every student. Now, as the teacher is having the sample copy with her which will be showed to the students for reference so that students can also make the same type of card with the same length and breadth.

Using that sample copy a number of cards with the same measurement, which can be used for different purposes, will be created by the students. Putting it differently, once the sample copy is created i.e. Class, it can be used as a reference in creating number of copies i.e. Object.

The sample copy cannot be used, as it is the copy, which will only show the exact measurement for reference of creating another card.

A collection of statements compiled together in order to execute an operation is defined as a Java method. For Example : the system executes a collection of statements to display a message while performing the System. Out print In method.

Given below is the technique of creating your own method with or without values, how to invoke a method with or without parameters, how to apply method abstraction in the program design and how to overload methods using same names.

❖ **Creating a Method :**

In general, a method has the following syntax :

```
Modifier return value type method name (list of parameters)
{
//Method Body;
}
```

A method definition consists of a method header and a method body. All the parts comprising a method are listed below :

1. **Modifiers –** It tells the compiler how the method is to be called and defines the access type of the method. The modifier is optional.
2. **Return Type –** The method may or may not return a value. In case it does, the return Value Type is the data type of the value which the method returns. Some methods perform the desired operations without returning a value. In this case, the return Value Type is the keyword void.
3. **Method Name –** This refers to the actual name of the method, which along with the parameter list comprises the method signature.
4. **Parameters –** A parameter is like a placeholder. When a method is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order and number of the parameters of a method. Parameters are optional; i.e., a method may contain no parameters."
5. **Method Body –** This is a collection of statements that define what the method does.

```
class display
{
 private int x;

 voidgetdata(int a)
 {
  x=a;
 }

 intreturndata( )
 {
  return x;
 }
}
```

In the above program, display is the name of the class with one data member, x and member functions, get data ( ) and return data ( ).

❖ **Creating Objects :**

Once the above code is written, a class with display name gets created along with the specified data members and member functions. However, no more objects are created at this point of time. In order to create objects, the new operator is used. For Example :

```
class_name object_name=new classname( )
```

It can also be alternatively written as :

```
class_name object_name ;
Objectname=new classname ( );
```

The new operator dynamically allocates memory for an object and returns a reference to it. The default value of object data type is null.

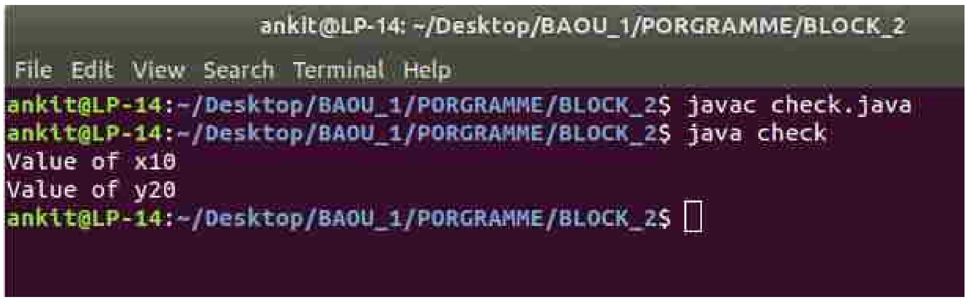
For Example :

```
display d;
//creates a reference
d=new display ( );
```

Now let us write a program to show the method of class creation and the way objects are created. For Example :

```
class display
{

 int x;
 int y;
 void getdata1(int a)
 {
  x=a;
 }

 int returndata1( )
 {
  return x;
 }

 void getdata2(int b)
 {
  y=b;
 }

 int returndata2( )
 {
  return y;
 }
}
```

```
class check
{

 public static void main (String[] args)
 {

  display c= new display();
  c.getdata1 (10);
  c.getdata2 (20);
  System.out.println ("Value of x" + c.returndata1 ());
  System.out.println ("Value of y" + c.returndata2 ());
 }
}
```


ankit@LP-14: ~/Desktop/BAOU_1/PORGRAMME/BLOCK_2
File Edit View Search Terminal Help

```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac check.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java check
Value of x10
Value of y20
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$
```

Figure 5.2 Output of Program

❑ **Check Your Progress – 1 :**

1. Define a class.
2. Write the syntax of a method.

....

....

....

....

....

5.3 Argument Passing :

Generally, there are two different methods of passing an argument to a function. These methods are :

a. Call by value

b. Call by reference

a. **Call by value –** In call by value method, the value of an argument is copied to the formal parameter. That is, the changes made in the actual argument are not reflected into the formal parameter.

Let us consider an Example : to illustrate the same :

```
//Call by Value Method

class check

{

inta,b;

void add(int x, int y)

{

a=x+10;

b=y+20;

}

}

class display

{

public static void main(String args [])

{

check c= new check ( );

int a=15, b=30;

System.out.println ("a and b before call" + a + " " + b);

c.add(a,b);

System.out.println ("a and b after call" + a + " " + b);

}

}
```

The output of above program will be :

```
a and b before call 15 30
a and b after call 15 30
```

b. **Call by Reference –** In the Call by Reference method, the changes made to the actual argument are also reflected in the formal argument. Let us take an Example : to see the illustration of the same :

```
//Call by Reference

class check

{

int a, b;

check (inti, int j)

{

a=i;
b=j;
}

//pass an object

void add (check c)
{

c.a+=5;
c.b*=4;
}

}

class display

{

public static void main (String args [])

{
check c= new check (4, 8);
System.out.println ("c.a and c.b before call : "+ c.a + " "+ c.b); c.add(c);
System.out.println ("c.a and c.b after call : "+ c.a + " " + c.b);
}

}
```

The output of the above program will be :

```
c.a and c.b before call : 4,8
c.a and c.b after call : 9,32
```

Notice the output of the above program, the values of a and b before and after the function call are different as the reference of these variables are passed to formal parameters. Hence, the changes made in actual parameters are directly reflected in formal parameters.

❑ **Check Your Progress – 2 :**

1. Explain the call by value method.
2. Write a note on the call by reference method.

....

....

....

....

....

5.4 Constructors :

Till now whatever we have studied, we have seen that if you want to initialize a variable it has to be done at the time of object creation. But it can be a tedious job to initialize all the variables in a class each time an object is created.

In the above program (at 5.3.a), the add( ) function is used to initialize the variables a and b, but, what if, the programmer forgets to call this add( ) function in the main( ) function. Then, it may happen that variables a and b can take garbage values.

So, in spite of giving the job of initialization to the programmer the entire responsibility of initialization is given to the compiler, which in turn is achieved by using constructors.

The constructors are not called in the main ( ) function. They are automatically executed at the time of object creation. For Example :, In the above program (at 5.3.a) using default constructors can be written as :

❖ **Default Constructor :**

The default constructors are those constructors which do not accept any parameters/arguments. These constructors are automatically executed at the time of object creation.

Let us take an Example : to illustrate the same :

```
Class rectangle
{
double length;
double breadth;
//Constructor for rectangle
rectangle ( )
{
System.out.println ("Constructing Rectangle");
length= 5.0;
breadth= 8.0;
}
double area( )
{
return length * breadth;
}
}

Class demo
{
public static void main (String args[])
{
//creating objects
rectangle r1 = new rectangle ( );
double result;
//get area of rectangle
result= r1.area( );
System.out.println ("Area is" + result);
//get area of second box
}
}
```

The output of above program is given below :

```
Constructing Rectangle
Area is 40
Area is 40
```

Notice the execution of the above program. At the time of object creation, the default constructors get executed, which can be visualized by the statement "Constructing Rectangle" which gets displayed when the object is created.

❖ **Parameterised Constructors :**

The parameterised constructors are those constructors which accept parameters/arguments. At the time of object creation the values of these arguments/parameters are passed from main ( ) function.

Consider the given Example :

```
class rectangle
{
double length;
double breadth;
//Constructor for rectangle
rectangle (double x, double y)
{
length =x;
breadth =y;
}
//compute and return area
double area ( )
{
return length * breadth;
}

}

class demo
{
public static void main (String args[])
{
//declare, allocate and initialize rectangle objects
rectangle r1= new rectangle(3,5);
rectangle r2=new rectangle (10,20);
double result;
//get area of first box
result=r1.area ( );
System.out.println ("Area is" + result);
//get area of second box
result=r2.area ( );
System.out.println ("Area is" + result);
}
}The output of above program will be :
Area is 15.0
Area is 200.0
```

In the above program, the values to the parameterised constructors are passed at the time of object creation in main function.

❑ **Check Your Progress – 3 :**

1. Write a note on default constructor.
2. Write an Example : for parameterised constructor.

....

....

....

....

....

5.5 The 'This' Keyword :

The 'this' keyword is used when a function will need to refer to the object which invoked it. The 'this' keyword can be used inside any method to refer to the current object. That is, it is always a reference to the object on which the method was invoked. Let us consider an Example : to illustrate the same :

```
Rectangle (double length, double breadth)
{
this.length=length;
this.breadth=breadth;
}
```

The use of this keyword is redundant but perfectly correct. Inside rectangle ( ), this will always refer to the invoking object. When a local variable has the same name as an instance variable, the local variable hides the instance variable. That is why length and width were not used as the name of the parameters to the rectangle ( ) constructor inside the rectangle class. If we would have used in that way, then breadth would have referred to the formal parameter hiding the instance variable breadth

❑ **Check Your Progress – 4 :**

1. Explain the use of this keyword.
2. Where can this keyword be used ?

....

....

....

....

....

5.6 The Finalize ( ) Method :

Till now, we have studied about the process of object creation. When objects are created using new keyword, the matching constructor is called automatically, where we initialize members of the class. This process is called initialization.

Sometimes an object needs to perform some operation when it is destroyed. For Example :, if an object is holding some non–Java resource such as a file handle or window character font, then you might want to make sure these resources are freed before an object is destroyed. To handle such situations, Java provides a mechanism called finalization.

Using the finalization keyword, you can define specific actions which will occur when an object is just to be reclaimed by the garbage collector. To add a finalizer to a class, simply define the finalise ( ) method, the Java run time calls that method whenever it is about to recycle an object of that class.

```
protected void finalize ( )
{

//finalization code here
}
```

As you know that Java's garbage collector runs in its own thread, it will run transparently alongside the execution of the program. If the programmer explicitly wants to request a garbage collection at some point of time, then it can be done by invoking System.gc( ) or Runtime.gc( ), which will fire off garbage collection at that time

❑ **Check Your Progress – 5 :**

1. Explain initialisation.
2. Write how to add a Final class.

....

....

....

....

....

3. A Class is also called as a ________ Factory.

   (A) Object (B) Method (C) Variable (D) Java

4. ________ tells the compiler how the method is to be called and defines the access type of the method.

   (A) Java (B) Class (C) Object (D) Modifiers

5. In ________ method, the value of an argument is copied to the formal parameter.

   (A) Static (B) call by value

   (C) call by reference (D) General

6. The ________ constructors are those constructors which do not accept any parameters/arguments.

   (A) Private (B) default (C) parameterised (D) Local

7. The ________ keyword is used when a function will need to refer to the object which invoked it..

   (A) my (B) this (C) these (D) those

8. ________ is automatically executed at the time of object creation.

   (A) Method (B) Constructor (C) Class (D) Variable

9. The 'this' keyword can be used inside any method to refer to the current object.

   (A) True (B) False

10. A class is nothing but a blueprint or a template for creating different objects.

    (A) True (B) False

11. In the Call by Reference method, the changes made to the actual argument are also reflected in the formal argument.

    (A) True (B) False

12. When objects are created using new keyword, the matching constructor is called automatically.

    (A) True (B) False

5.7 Let Us Sum Up :

In this Unit we have learned Object Oriented Concepts we can say that a object is an identifiable entity with some characteristic features and behavior. Anything which has some properties and performs some behavior is called an object. Keeping mind the programming for user's application is said as object oriented programming. In mean time we understood Classes which is nothing but a Collection of similar types of objects is called Class. A Class is also called as an Object Factory as once the class is created we can create as many objects as we wish using that class.

There is something called Java Method .A Java method is a collection of statements that are grouped together to perform an operation. Creating a Method

In general, a method has the following syntax :

```
Modifier, returnvaluetype, methodname (list of parameters)
{
//Method Body;
}
```

A method definition consists of a method header and a method body like Modifiers, Return Type, and Method Name, Parameters, Method Body.Creating Objects,

Further we have learned that in general there are two different methods of passing an argument to a function. These methods are i) Call by value, ii. Call by reference. Call by value : In call by value method, the value of an argument is copied to the formal parameter. That is, the changes made in the actual argument are not reflected into the formal parameter, and Call by Reference, in Call by Reference method, the changes made to the actual argument are also reflected in the formal argument.

It is also understood that the constructors are not called in the main ( ) function. They are automatically executed at the time of object creation. For Example : the above program using default constructors can be written as DEFAULT CONSTRUCTOR. The default constructors are those constructors which do not accept any parameters/arguments. These constructors are automatically executed at the time of object creation. Another thing is related to Parameterised Constructors. The parameterised constructors are those constructors which accept parameters/arguments. At the time of object creation the values of these arguments/parameters are passed from main ( ) function.

There is learning about the 'this' keyword is used when a function will need to refer to the object which invoked it. The 'this' keyword can be used inside any method to refer to the current object. At the end of this Unit we came to know that the use of this keyword is redundant but perfectly correct. Inside rectangle ( ), this will always refer to the invoking object. When a local variable has the same name as an instance variable, the local variable hides the instance variable. That is why length and width were not used as the name of the parameters to the rectangle ( ) constructor inside the rectangle class. If we would have used in that way, then breadth would have referred to the formal parameter hiding the instance variable breadth

5.8 Suggested Answer for Check Your Progress :

- **Check Your Progress 1 :**

  See Section 5.2

- **Check Your Progress 2 :**

  See Section 5.3

- **Check Your Progress 3 :**

  See Section 5.4

- **Check Your Progress 4 :**

  See Section 5.5

❑ **Check Your Progress 5 :**

1 : See Section 5.6		**2 :** See Section 5.6		
3 : A	**4 :** D	**5 :** B	**6 :** B	**7 :** B
8 : B	**9 :** A	**10 :** A	**11 :** A	**12 :** A

5.9 Glossary :

1. **Modifiers** – is one which communicates to the compiler how to call the method. This defines the access type of the method.
2. **Return Type** – A method may return a value. The return Value Type is the keyword void.
3. **Method Name** – This is the actual name of the method. The method name and the parameter list together constitute the method signature.
4. **Parameters** – A parameter is like a placeholder. When a method is invoked, you pass a value to the parameter

5.10 Assignment :

1. Write the steps to create objects.
2. Explain the process of creating a method.

5.11 Activities :

Define a class student with data members studied, name, address, age and marks of 3 subjects. Provide the necessary constructors and methods along with getmarks( ) method and display the total and percentage marks with complete information about student.

5.12 Case Study :

Define a class Rectangle with its length and breadth. Provide appropriate constructor(s), which gives facility of constructing Rectangle object with default values of length and breadth as 0 or passing value of length and breadth externally to constructor. Provide methods to calculate area and method display to display all information of Rectangle. Design different class Test Rectangle class in separate source file, which will contain the main function. From this main function, create an object which is a Rectangle and call the methods area and display.

5.13 Further Reading :

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000
2. Java 2, The Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999
3. Programming with Java, Ed. 2,E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000
4. The Java Tutorial, Ed. 2, 2 volumes, Mary Campione and Kathy Walrath, Addison Wesley Longmans, 1998
5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele & Gilad Bracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000
6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition, 2000

Unit 06

LANGUAGE FEATURES

UNIT STRUCTURE

6.0 Learning Objectives :

After learning this unit, you will be able to :

- Static Keyword
- Abstract Classes
- Interfaces
- Packages
- Access Protection

6.1 Introduction :

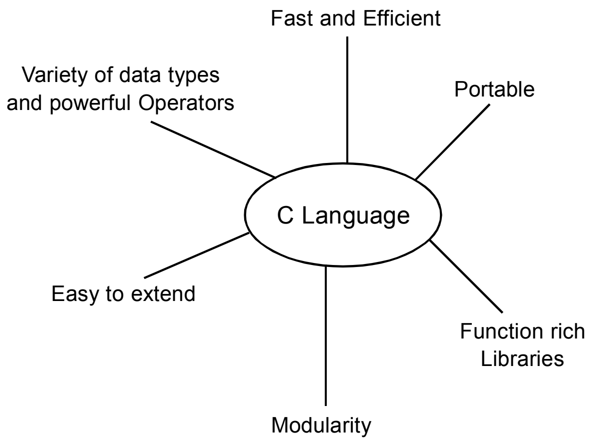


Figure 6.1 Feature of C language

The development of Java has been a compilation of the best points of various programming languages such as C and C++. Java therefore utilizes algorithms and methodologies that are already proven. The Java environment automatically tackles tasks which are prone to errors such as pointers and memory management rather than the programmer taking the initiative.

Since Java is primarily a derivative of C++ that most programmers are conversant with, it implies that Java has a familiar feel rendering it easy to use. The Java language supports many high–performance features such as multithreading, just–in–time compiling and native code usage.

6.2 Static Keyword :

When an object is created or, primitive type variable or method is called, the memory for that object, variable or method is set aside.

The different objects, variables and methods occupy different areas of memory when created/called. In some cases, we would like to have multiple objects, variables or methods which occupy the same area of memory (in effect just having the one instance of that variable or method). The above can be achieved by using the static keyword; it is possible to have static methods and variables.

In Java, global variables are not allowed. In order to do the same, the instance variable in the class can be declared static. The effect of doing this is that when we create multiple objects of that class, every object shares the same instance variable that was declared to be static.

To make an instance variable static, we simply precede the declaration with the static keyword :

```
public static int Instance_variable = 0
```

In effect, what we are really doing is saying that this instance variable, no matter how many objects are created should always reside in the same memory location regardless of the object. This then stimulates a 'global variable' of sorts.

We usually make a variable declared to be final, static as well since it makes sense to only have the one instance of a constant. The static instance variables are also called as class variables.

Outside of the class in which they are defined, static methods and variables can be used independently of any object. In order to do so, you only need to specify the name of the class followed by the dot operator.

❑ Check Your Progress – 1 :

1. Explain how to make an instance variable static.
2. What are static instance variables also called as ?

....

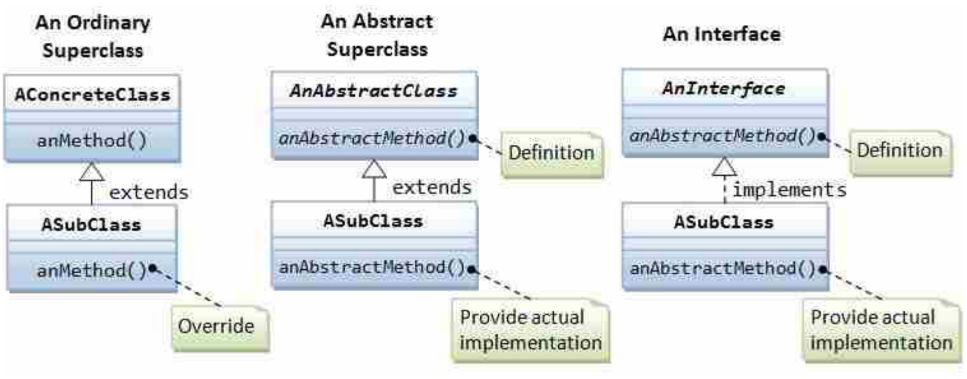
....

....

....

....

6.3 Using Abstract Classes :



There are often situations where you want to determine a superclass, which without providing a complete implementation of every method declares the structure of an abstraction. That is, many a times you'll want to create a superclass that only defines a generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details.

The abstract keyword can be used with :

1. A class
2. A method

❖ Abstract Method :

In a method declaration, abstract indicates that the implementation will be in subclass. Since these methods do not have an implementation specified in the superclass they are sometimes cited as subclasser responsibility. Hence, a subclass cannot use the version defined in the superclass, it must override them. To declare an abstract method, use this general form :

abstract type Method_Name (parameter–list);

No method body is present as specified above.

❖ Abstract Class :

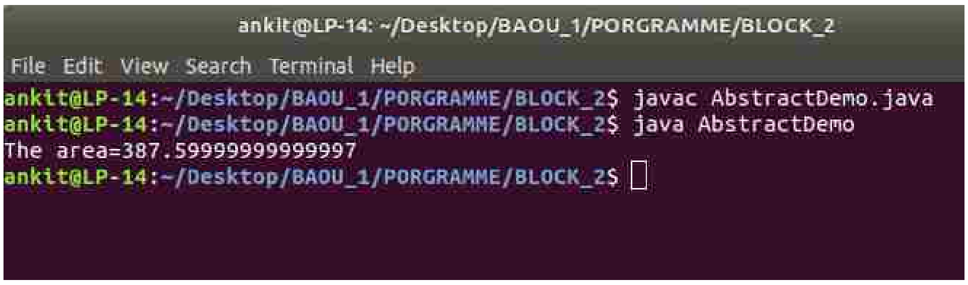
A class that is declared abstract is defined as an abstract class. The class need not necessarily include abstract methods and can be subclassed. Abstract classes cannot be instantiated.

In order to declare a class as abstract, you have to use the abstract keyword before the class keyword at the beginning of the class declaration. There can be no objects of the abstract class, that is, an abstract class cannot be directly instantiated with the new operator.

Any derived class that does not implement all abstract methods of its superclass must be declared abstract. Let us take an Example : to understand this concept in more detail.

In the given program, the class Figure is declared as abstract because we don't want objects of this class to be created. Instead, this class should be subclassed. Notice that the method area ( ) is also abstract because we cannot define it in the Figure class. It is defined in the subclass –Rect

```
abstract class Figure
{

 protected double dim1, dim2;
 Figure(double dim1, double dim2)
 {
  this.dim1 =dim1;
  this.dim2 =dim2;
 }

 abstract double area(); //abstract method
}

class Rect extends Figure
{

 Rect(double l, double d)
 {
  super(l,d);
 }
 double area()
 {
  return (dim1 * dim2);
 }

}

public class AbstractDemo
{
 public static void main (String args [ ])
 {
  Rect r = new Rect(15.2, 25.5);
  System.out.println("The area=" + r.area ());
 }
}
```



```
ankit@LP-14: ~/Desktop/BAOU_1/PORGRAMME/BLOCK_2
File Edit View Search Terminal Help
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac AbstractDemo.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java AbstractDemo
The area=387.59999999999997
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ 
```

Figure 6.3 Output of Program

❑ **Check Your Progress – 2 :**

1. Where is the abstract keyword used ?
2. Write the general form of abstract method.

....

....

....

....

....

6.4 Interfaces :

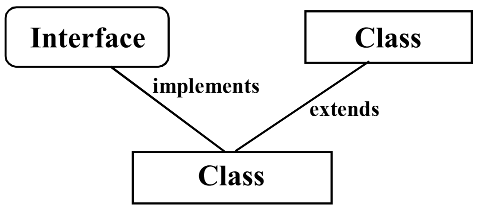


Figure 6.4 Interface

"A collection of abstract methods is an interface. Thus, be inheriting the abstract methods of an interface a class implements an interface."

"An interface is not a class. They are two different concepts but writing an interface is similar to a class. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements."

"Every method of the interface is defined in the class unless the class implementing the interface is abstract."

An interface is similar to a class in the following ways :

- The interface contains various methods.
- The name of the interface matches the name of the file and it is written with a .java extension.
- The bytecode of an interface appears in a .class file.
- An interface appears in packages and the bytecode file it corresponds to must appear in a directory structure matching its name.

However, an interface is different from a class in several ways, including :

- You cannot instantiate an interface.
- Constructors do not constitute an interface.
- All of the methods in an interface are abstract.

- An interface can only contain fields that are declared both static and final and it cannot contain instance fields.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

❖ **Declaring Interfaces :**

The interface keyword is used to declare an interface.

Encapsulation is defined as a barrier protecting and preventing the code and data from being randomly accessed by other code outside the class. The access is tightly controlled by an interface.

The main benefit of encapsulation is the ability to modify our implemented code without breaking the code of others who use our code. With this feature Encapsulation gives maintainability, flexibility and extensibility to our code.

Example :

Let us look at an Example : that depicts encapsulation :

```
/* File name : NameOfInterface.java */
import java.lang.*
//Any number of import statements
public interface NameOfInterface
{
   //Any number of final, static fields
   //Any number of abstract method declarations\
}
```

Interfaces have the following properties :

- While declaring an interface you do not need to use the abstract keyword since the interface is implicitly abstract.
- The abstract keyword is not needed as each method in an interface is implicitly abstract.
- Methods in an interface are implicitly public.

❖ **Implementing Interfaces :**

The process of a class implementing an interface can be seen as the class signing a contract, complying to carry out certain behaviors of the interface. In case a class fails to carry out these behaviors, the class must declare itself abstract.

In a class the implements keyword is used to implement the interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

"When you define overriding methods in interfaces, the following rules are to be followed :

- Checked exceptions should not be declared on implementation methods other than the ones declared by the interface method or subclasses of those declared by the interface method.
- When overriding methods, you must maintain the signature of the interface method and also the same return type or subtype.

- Interface methods do not have to be implanted if in case an implementation class itself is abstract.
- While implementing interfaces, there are several rules :
  - o A class can implement more than one interface at a time.
  - o A class can extend only one class but implement many interfaces.
  - o An interface itself can extend another interface. An interface cannot extend another interface."

❖ **Extending Interfaces :**

Just as a class can extend another class, an interface can extend another interface as well. The extends keyword is used to extend an interface and the child interface inherits the methods of the parent interface.

"The following Sports interface is extended by Hockey and Football interfaces.

```
//Filename : Sports.java
public interface Sports
{
   public void setHomeTeam(String name)
   public void setVisitingTeam(String name)
}

//Filename : Football.java
public interface Football extends Sports
{
   public void homeTeamScored(int points)
   public void visitingTeamScored(int points)
   public void endOfQuarter(int quarter)
}
//Filename : Hockey.java
public interface Hockey extends Sports
{
   public void homeGoalScored()
   public void visitingGoalScored()
   public void endOfPeriod(int period)
   public void overtimePeriod(intot)
}
```

The Hockey interface has four methods but it inherits two from Sports; thus, a class that implements Hockey needs to implement all six methods. Similarly, a class that implements Football needs to define the three methods from Football and the two methods from Sports."

❑ **Check Your Progress – 3 :**

1. Explain in what ways is an interface similar to a class.
2. Write the rules for implementing interfaces.

....

....

....

....

....

❑ **Check Your Progress – 4 :**

1. Write a note on abstract class number.

....

....

....

....

....

6.5 Packages :

Java uses packages to avoid naming conflicts, to ease the searching and usage of interfaces, classes, annotations and enumerations and to control access.

Packages are a collection or group of related types of (classes, interfaces, enumerations and annotations) providing access protection and name space management.

Some of the existing packages in Java are :

- java.lang – bundles the fundamental classes
- java.io – classes for input , output functions are bundled in this package

Programmers can bundle up a group of classes/interfaces in order to define their own packages. It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, annotations are related.

There are to be no conflicts with names in various other packages since a package creates a new namespace. With the help of packages, providing access control and locating related classes can be done with ease.

❖ **Creating a Package :**

You have to select a name for the package and put a package statement with that very name at the top of every source file that contains the classes, interfaces, enumerations and annotation types that you want to include in the package when you are creating it.

The first line in the source file must be the package statement. Each source file can have only one package statement which shall apply to all types in the file.

The class, interfaces, enumerations and annotation types are put into an unnamed package if a package statement is not used.

Example :

Let us look at an Example : that creates a package called animals. It is a common practice to use lowercased names of packages to avoid any conflicts with the names of classes, interfaces.

Put an interface in the package animals :

```
/* File name : Animal.java */
package animals

interface Animal {
public void eat()
public void travel()
}
```

Now put an implementation in the same package *animals* :

```
package animals;

/* File name : MammalInt.java */
public class MammalInt implements Animal{

   public void eat(){
System.out.println("Mammal eats")
   }

   public void travel(){
System.out.println("Mammal travels")
   }

   public intnoOfLegs(){
      return 0
   }

   public static void main(String args[]){
MammalInt m = new MammalInt();
m.eat();
m.travel();
   }
}
```

❖ **The import Keyword :**

If a class wants to use another class in the same package, the package name does not need to be used. Classes in the same package find each other without any special syntax.

Example :

Here a class named Boss is added to the payroll package that already contains Employee. The Boss can then refer to the Employee class without using the payroll prefix, as demonstrated by the following Boss class.

```
package payroll;

public class Boss
{
public void payEmployee(Employee e)
{
e.mailCheck();
}
}
```

- The package can be imported using the import keyword and the wild card (*) character. For Example :,

  import payroll.*

- The class itself can be imported using the import keyword. For Example :

  import payroll.Employee;

Note : A class file can contain any number of import statements. The import statements must appear after the package statement and before the class declaration.

❖ **The Directory Structure of Packages :**

When a class is placed in a package, the following results are concluded :

- As stated in the previous section, the name of the package becomes a part of that of the classes' name.
- The name of the package must match the directory structure where the corresponding bytecode resides.

❑ **Check Your Progress – 5 :**

1. Name the existing packages in Java.
2. What are the results when a class is placed in a package ?.

....

....

....

....

....

6.6 Access Protection :

Packages add another dimension to access control, they act as containers for classes, other subordinate packages, data and code. The class is Java's smallest unit of abstraction, due to the interplay between classes and packages, Java addresses four categories of visibility for class members, which are mentioned below :

- Subclasses in the same package.
- Non–subclasses in the same package.
- Subclasses in different packages.
- Classes that are neither in the same package nor subclasses.

Table 6.1 : Class Member Access

Access rights for the different elements

class \ have access to	**private elements**	**default elements (*no modifier*)**	**protected elements**	**public elements**
own class (**Base**)	**yes**	**yes**	**yes**	**yes**
subclass - same package (**SubA**)	**no**	**yes**	**yes**	**yes**
class - same package (**AnotherA**)	**no**	**yes**	**yes**	**yes**
subclass - another package (**SubB**)	**no**	**no**	**yes/no +**	**yes**
class - another package (**AnotherB**)	**no**	**no**	**yes**	**yes**

The three access specifies, private, public and protected, provide a variety of ways to produce the many levels of access required by these categories.

Anything declared public can be accessed from anywhere, whereas anything declared private cannot be seen outside of its class. When a member does not have an explicit access specification, it is visible to subclasses as well as to other classes in the same package, which is the default access.

An element is declared protected if you want to allow an element to be seen outside your current package but only to classes that subclass your class directly.

Table 6.1 is applicable only to members of classes. A class has only two possible access levels : default and public.

Let us consider an Example : to illustrate the above concepts :

```
packagepackageA;

public class Base
{

 public String publicStr = "publicString";
 protected String protectedStr = "protectedString";
 String defaultStr = "defaultString";
 private String privateStr = "privateString";

 public void print()
 {

  System.out.println("packageA.Base has access to");
  System.out.println(" " + publicStr);
  System.out.println(" " + protectedStr);
  System.out.println(" " + defaultStr);
  System.out.println(" " + privateStr);
```

```
    Base b = new Base(); // -- other Base instance
    System.out.println(" b." + b.publicStr);
    System.out.println(" b." + b.protectedStr);
    System.out.println(" b." + b.defaultStr);
    System.out.println(" b." + b.privateStr);
  }
}
-------------------------------------------------------------------------------------------
packagepackageB;

importpackageA.Base;

public class SubB extends Base
{
  public void print()
  {
    System.out.println("packageB.SubB has access to");
    System.out.println(" " + publicStr + " (inherited from Base)");

   //-- protectedStr is inherited element -> accessible System.out.println("
" + protectedStr + " (inherited from Base)");
    //-- not accessible
    //--System.out.println(defaultStr);
    //--System.out.println(privateStr);
    Base b = new Base(); // -- other Base instance System.out.println("
b." + b.publicStr)
   //-- protected element, which belongs to other object -> not accessible
    //--System.out.println(b.protectedStr);
    //-- not accessible
    //--System.out.println(b.defaultStr);
    //--System.out.println(b.privateStr);
  }
}
-------------------------------------------------------------------------------------------
import packageA.*;

import packageB.*;

// -- testing class
```

```
public class TestProtection
{
 public static void main(String[] args)
 {
  //-- all classes are public, so class TestProtection
  //-- has access to all of them
  new Base().print();
  newSubA().print();
  newAnotherA().print();
  newSubB().print();
  newAnotherB().print();
 }
}
```

❖ **Types of Variables :**

There are three kinds of variables in Java :

1. Local variables
2. Instance variables
3. Class/static variables

❖ **Local variables :**

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables so local variables should be declared and an initial value should be assigned before the first use.

❖ **Instance variables :**

- Instance variables are declared in a class but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.

- The instance variables are visible for all methods, constructors and block in the class. Normally it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers the default value is 0, for Booleans it is false and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods and different class (when instance variables are given accessibility) they should be called using the fully qualified name. Object Reference. Variable Name.

❖ **Class/static variables :**

- Class variables also known as static variables are declared with the static keyword in a class but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final and static. Constant variables never change from their initial value.
- Static variables are stored in static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables. For numbers the default value is 0, for Booleans it is false and for object references it is null. Values can be assigned during the declaration or within the constructor. Additionally values can be assigned in special static initialiser blocks.
- Static variables can be accessed by calling with the class name. Class Name. Variable Name.
- When declaring class variables as public static final, then variables names (constants) are all in upper case. If the static variables are not public and final the naming syntax is the same as instance and local variables.

Note : If the variables are accessed from an outside class, the constant should be accessed as Employee. Department

❑ **Check Your Progress – 6 :**

1. List the four categories of visibility for class members.
2. Write a note on instance variables ?

....

....

....

....

....

3. Java compiler convert source code in to ________ code.

   (A) Binary (B) Machine (C) English (D) Byte

4. JVM Stand for ________.

   (A) Java Virtual Machine (B) Java Version Machine

   (C) Java Virtual Mode (D) Java Version Mode

5. No method body is present as ________ method.

   (A) Final (B) Public (C) Private (D) Abstract

6. "A collection of abstract methods is called ________.

   (A) Package (B) Interface (C) Class (D) Abstract class

7. The ________ keyword is used to extend an interface

   (A) Extends (B) inherit (C) final (D) depends

8. ________ are a collection or group of related types of classes, interfaces, enumerations and annotations.

   (A) Packages (B) class (C) Interface (D) Namespace

9. Abstract keyword can be used with class only.

   (A) True (B) False

10. The Abstract class need not necessarily include abstract methods.

    (A) True (B) False

11. Interface is not a class.

    (A) True (B) False

12. All of the methods in an interface are abstract.

    (A) True (B) False

6.7 Let Us Sum Up :

When an object is created or, primitive type variable or method is called, the memory for that object, variable or method is set aside.

The different objects, variables and methods occupy different areas of memory when created/called. In some cases, we would like to have multiple objects, variables or methods which occupy the same area of memory (in effect just having the one instance of that variable or method). The above can be achieved by using the static keyword; it is possible to have static methods and variables.

In Java, global variables are not allowed. In order to do the same, the instance variable in the class can be declared static. The effect of doing this is that when we create multiple objects of that class, every object shares the same instance variable that was declared to be static.

Sometimes there are situations in which you will want to define a superclass, which declares the structure of a given abstraction without providing a complete implementation of every method. That is, many a times you'll want to create a superclass that only defines a generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details. The abstract keyword can be used with : a) A class, b) A method

An interface is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. An interface is not a class. Writing an interface is similar to writing a class but they are two different concepts. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements. Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

There is also learning about an interface is similar to a class in the several ways : However, an interface is different from a class in several ways. Further we learned about Declaring Interfaces, in this the interface keyword is used to declare an interface. Here is a simple Example : to declare an interface. Next thing which we understood is encapsulation can be described as a protective barrier that prevents the code and data being randomly accessed by other code defined outside the class. Access to the data and code is tightly controlled by an interface.

There is also learning related to Packages are used in Java in–order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier. Packages add another dimension to access control, they act as containers for classes and other subordinate packages. Classes act as containers for data and code. The class is Java's smallest unit of abstraction, due to the interplay between classes and packages, Java addresses four categories of visibility for class members, which are 1) Subclasses in the same package. 2) Non–subclasses in the same package. 3) Subclasses in different packages. 4) Classes that are neither in the same package nor subclasses.

6.8 Suggested Answer for Check Your Progress :

❑ **Check Your Progress 1 :**

See Section 6.2

❑ **Check Your Progress 2 :**

See Section 6.3

❑ **Check Your Progress 3 :**

See Section 6.4

❑ **Check Your Progress 4 :**

See Section 6.5

❑ **Check Your Progress 5 :**

See Section 6.5

❑ **Check Your Progress 6 :**

1 : See Section 6.7		**2 :** See Section 6.7		
3 : D	**4 :** A	**5 :** D	**6 :** B	**7 :** A
8 : A	**9 :** A	**10 :** A	**11 :** A	**12 :** A

6.9 Glossary :

1. Interface – An interface is a collection of abstract methods.
2. Instance variables – are declared variables in a class but outside a method, constructor or any block.
3. Local variables – Local variables are declared in methods, constructors, or blocks.

6.10 Assignment :

Write a note on Java technology.

6.11 Activities :

Write any two programs to show the use of Interfaces

6.12 Case Study :

Explain Java programming environment with the help of diagram

6.13 Further Reading :

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000.
2. Java 2, the Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999.
3. Programming with Java, Ed. 2, E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000.
4. The Java Tutorial, Ed. 2, 2 volumes, Mary Campione and Kathy Walrath, Addison Wesley Longmans, 1998.
5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele & Gilad Bracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000.
6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition, 2000

Unit 07

WRAPPER CLASSES

UNIT STRUCTURE

7.0 Learning Objectives :

After learning this unit, you will be able to understand the use of :

- Byte Class
- Short Class
- Integer Class
- Long Class
- Float Class
- Double Class
- Boolean Class
- Character Class
- String Class

7.1 Introduction :

Java framework provides the Java.lang package. The java.lang contains several classes and interfaces. Java also provide the basic data types like bytes, short, int, long, float, double, char and Boolean. These all data types are used using variables. Sometimes programmer need to use the same data type as an objects. So Java provides object wrappers mechanism, this mechanism transformed the basic type in to object and become immutable. The object of the concern type can help to pass them by reference to methods instead of passing by value. The wrapper mechanism is as follow :

Integer int_obj = new Integer (35);

Here int_obj object encapsulates (wrapper) the integer value 35. So here the Integer class is known as a wrapper type.

7.2 Number Class :

Java provide abstract class "Number" under the java.lang package. The Number class contain six concrete subclasses to wrap the basic types. All the six classes provides several methods which can be used for I/O operation and conversion from one form the other form.

The six subclasses are as follow. :

- Byte
- Short
- Integer
- Long
- Double
- Float

We will discuss the methods of each class latter in this unit. Here bellow table 7.1 show the common methods that are available in all six classes.

Table 7.1 : Common Methods in all subclasses of Number Class

Method	Description
byte byteValue()	Return the byte value of the invoking object
short shortValue()	Return the short value of the invoking object
int intValue()	Return the integer value of the invoking object
long longValue()	Return the long value of the invoking object
float floatValue()	Return the float value of the invoking object
double doubleValue()	Return the double value of the invoking object
String toString()	Return the string value of the invoking object
int hashCode()	Return the hashcode value of the invoking object

7.3 Byte Class :

The Byte class is a wrapper class for the byte data type. The constructors for Byte class are as follow :

❖ **Constructors :**

Byte (byte num);

Byte(string str);

Here num is byte type and str is string representation of a byte.

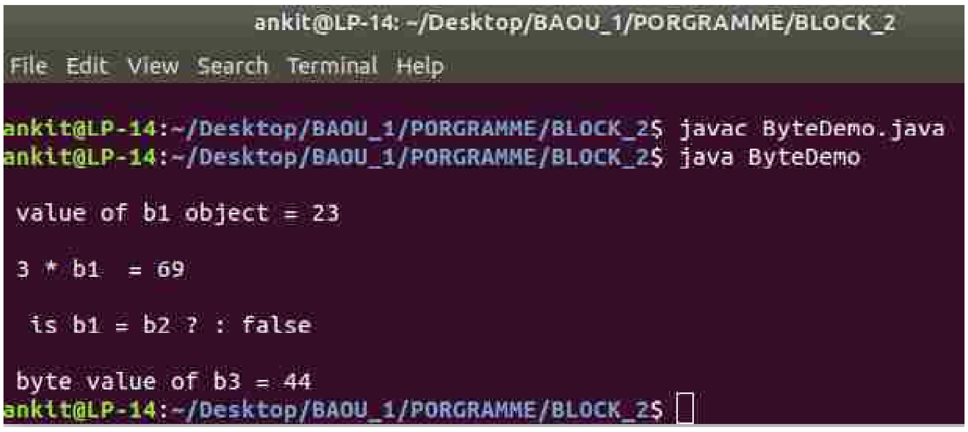
The bellow table 7.2 gives a brief description of methods available under Byte Class :

Table 7.2 : Methods defined in Byte Class

Method	Description
byte byteValue( )	This method return the value of the invoking object as a byte.
int compareTo(Byte byte)	This method compare the numerical value of the invoking object with that of byte. It will return 0 if the values are equal. It will return a negative value if the invoking object has a lower value. It will return a positive value if the invoking object has a greater value.
int compareTo(Object obj)	To operate same as compareto(Byte)method, if obj is of class Byte Otherwise, this method will throw a ClassCastException.
double doubleValue( )	This method return the value of the invoking object as a double.
boolean equals(Object Obj)	This method return true if the invoking Byte object is equivalent to Obj. Otherwise, it return false.
float floatValue( )	This method return the value of the invoking object as a float.
int intValue( )	This method return the value of the invoking object as an int.
long longValue( )	This method return the value of the invoking object as a long.
static byte parseByte(String s) throws NumberFormat Exception	This method return the byte equivalent of the number contained in the string specified by s. This method will used radix 10.
short shortValue( )	This method return the value of the invoking object as a short.
static String toString(byte number)	This method return a string that contains the decimal equivalent of number.
static Byte valueOf(String s) throws NumberFormat Exception	This method return a Byte object that contains the value specified by the string in s.
static Byte valueOf(String s, int radix) throws NumberFormat Exception	This method return a Byte object that contains the value specified by the string in s using the specified radix.

The following program show the use of some methods defined in Byte class :

```
class ByteDemo
{
 public static void main(String args[])
 {
  Byte b1 = new Byte((byte)23);
  Byte b2 = new Byte ("80");

  System.out.println("\n value of b1 object = " + b1);
  System.out.println("\n 3 * b1  = " + 3 * b1.byteValue());
  System.out.println("\n  is b1 = b2 ? : " + b1.equals(b2));
  Byte b3 = Byte.parseByte("44");
  System.out.println("\n byte value of b3 = " + b3);
 }
}
```


ankit@LP-14: ~/Desktop/BAOU_1/PORGRAMME/BLOCK_2
File Edit View Search Terminal Help

```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac ByteDemo.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java ByteDemo

 value of b1 object = 23

 3 * b1  = 69

  is b1 = b2 ? : false

 byte value of b3 = 44
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$
```

Figure 7.1 Output of Program

❑ **Check Your Progress – 1 :**

1. Explain Byte wrapper Class.

....

....

7.4 Short Class :

The Short class is a wrapper class for the short data type. The constructors for short class are as follow :

❖ **Constructors :**

Short(short num);

Short(string str);

Here num is short type and str is string representation of a short.

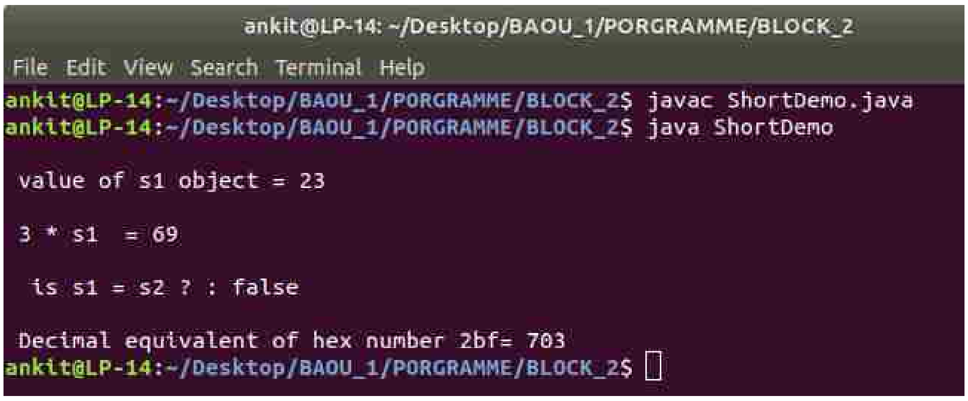
The bellow table 7.3 gives a brief description of methods available under Short Class :

Table 7.3 : Methods defined in Short Class

Method	Description
byte byteValue( )	This method return the value of the invoking object as a byte.
int compareTo(Short obj)	This method compare the numerical value of the invoking object with that of byte. It will return 0 if the values are equal. It will return a negative value if the invoking object has a lower value.
int compareTo(Object obj)	This method operate same as compareTo (Byte)method, if obj is of class Short Otherwise, this method will throw a ClassCast Exception.
double doubleValue( )	This method return the value of the invoking object as a double.
boolean equals(Object Obj)	This method return true if the invoking Integer object is equivalent to Obj. Otherwise, it return false.
float floatValue( )	This method return the value of the invoking object as a float.
int intValue( )	This method return the value of the invoking object as an int.
long longValue( )	This method return the value of the invoking object as a long.
static byte parseByte(String str) throws NumberFormat Exception	This method return the short equivalent of the number contained in the string specified by str using radix 10.
short shortValue( )	This method return the value of the invoking object as a short.
static toString(short number)	This method return a string that contains the decimal equivalent of the invoking obj.
static String toString(short number)	This method return a string that contains the decimal equivalent of number.
static Short valueOf(String s) throws NumberFormat Exception	This method return a Short object that contains the value specified by the string in s. This method will use radix 10.
static Short valueOf(String s, int radix) throws NumberFormatException	This method return a Short object that contains the value specified by the string in s. This method will use the radix specified by the user.

The following program show the use of some methods defined in Short class :

```
class ShortDemo
{
 public static void main(String args[])
 {
  Short s1 = new Short((short)23);
  Short s2 = new Short("80");

  System.out.println("\n value of s1 object = " + s1);
  System.out.println("\n 3 * s1  = " + 3 * s1.shortValue());
  System.out.println("\n  is s1 = s2 ? : " + s1.equals(s2));
  short s3 = Short.valueOf("2bf",16);
 System.out.println("\n Decimal equivalent of hex number 2bf= " + s3);
 }
}
```



```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac ShortDemo.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java ShortDemo

 value of s1 object = 23

 3 * s1  = 69

  is s1 = s2 ? : false

 Decimal equivalent of hex number 2bf= 703
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$
```

Figure 7.2 Output of Program

❑ **Check Your Progress – 2 :**

1. Explain Short wrapper Class.

....

....

7.5 Integer Class :

The Integer class is a wrapper class for the int data type. The constructors for Integer class are as follow :

❖ **Constructors :**

Integer(int num);

Integer(string str);

Here num is int type and str is string representation of a int.

The bellow table 7.4 gives a brief description of methods available under Integer Class :

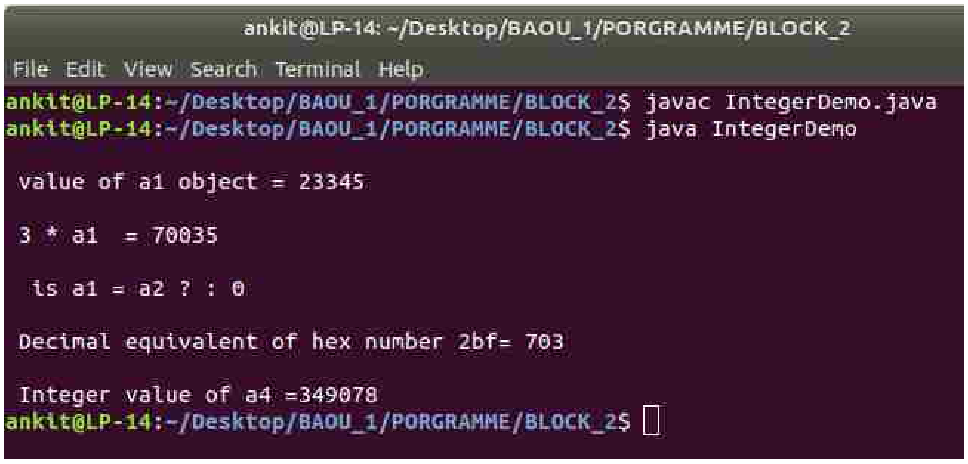
Table 7.4 : Methods defined in Interger Class

Method	Description
byte byteValue( )	This Method return the value of the invoking object as a byte.
int compareTo(Integer number)	Compare the numerical value of the invoking object with that of number. It will return 0 if the values are equal. It will return a negative value if the invoking object has a lower value. It will return a positive value if the invoking object has a greater value.
double doubleValue( )	This Method return the value of the invoking object as a double.
boolean equals(Object Obj)	This Method return true if the invoking Integer object is equivalent to Obj. Otherwise, it return false.
float floatValue( )	To return the value of the invoking object as a float.
int intValue( )	This Method return the value of the invoking object as an int.
long longValue( )	This Method return the value of the invoking object as a long.
static int parseByte(String s) throws NumberFormat Exception	This Method return the byte equivalent of the number contained in the string specified by s. This method will used radix 10.
short shortValue( )	This Method return the value of the invoking object as a short.
static toString(short number)	This Method return a string that contains the decimal equivalent of the invoking object.
static String toString(short number)	This Method return a string that contains the decimal equivalent of number.
static Short valueOf(String s) throws NumberFormat Exception	This Method return a Short object that contains the value specified by the string in s.

The following program show the use of some methods defined in Integer class :

```
class IntegerDemo
{
 public static void main(String args[])
 {
  Integer a1 = new Integer((23345));
```

```
    Integer a2 = new Integer("23345");

    System.out.println("\n value of a1 object = " + a1);
    System.out.println("\n 3 * a1  = " + 3 * a1.shortValue());
    System.out.println("\n  is a1 = a2 ? : " + a1.compareTo(a2));
    Integer a3 = Integer.valueOf("2bf",16);
   System.out.println("\n Decimal equivalent of hex number 2bf= " + a3);
    int a4 = Integer.parseInt("349078");
    System.out.println("\n Integer value of a4 =" + a4);
   }
  }
```



```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac IntegerDemo.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java IntegerDemo

 value of a1 object = 23345

 3 * a1  = 70035

  is a1 = a2 ? : 0

 Decimal equivalent of hex number 2bf= 703

 Integer value of a4 =349078
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$
```

Figure 7.3 Output of Program

❑ **Check Your Progress – 3 :**

1. Explain Integer wrapper Class.

   ....

   ....

7.6 Long Class :

The Long class is a wrapper class for the long data type. The constructors for Long class are as follow :

❖ **Constructors :**

Long (long num);

Long(string str);

Here num is long type and str is string representation of a long.

The bellow table 7.5 gives a brief description of methods available under Long Class :

Table 7.5 : Methods defined in Long Class

Method	Description
byte byteValue( )	This method return the value of the invoking object as a byte.
int compareTo(Long number)	Compare the numerical value of the invoking object with that of number. It will return 0 if the values are equal. It will return a negative value if the invoking object has a lower value. It will return a positive value if the invoking object has a greater value.
double doubleValue( )	This method return the value of the invoking object as a double.
boolean equals(Object Obj)	This method return true if the invoking long object is equivalent to Obj. Otherwise, it return false.
float floatValue( )	This method return the value of the invoking object as a float.
int intValue( )	This method return the value of the invoking object as an int.
long longValue( )	This method return the value of the invoking object as a long.
static long parseByte(String s) throws NumberFormat Exception	This method return the byte equivalent of the number contained in the string specified by s. This method will used radix 10.
short shortValue( )	This method return the value of the invoking object as a short.
static toString(long number)	This method return a string that contains the decimal equivalent of the invoking object.
static String toString(long number)	This method return a string that contains the decimal equivalent of number.
static Long valueOf(String s) throws NumberFormat Exception	This method return a Short object that contains the value specified by the string in s.

7.7 Float Class :

The Float class is a wrapper class for the float data type. The constructors for Float class are as follow :

❖ **Constructors :**

Float (float num);

Float(double num);

Float(String str);

Here num is float and double type respectively and str is string representation of a float.

The bellow table 7.6 gives a brief description of methods available under Float Class :

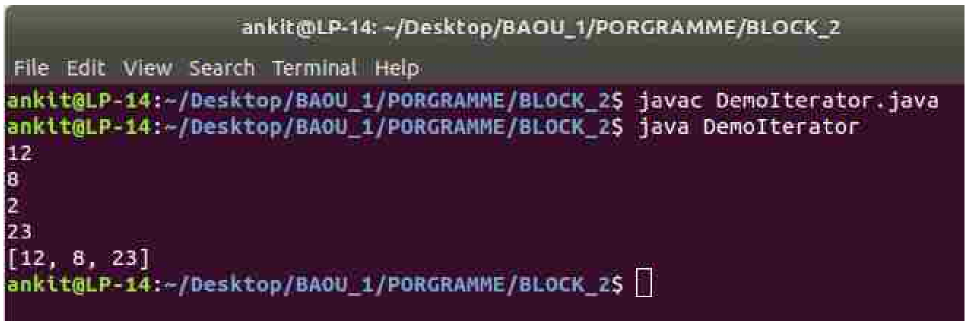
Table 7.6 : Methods defined in Float Class

Method	Description
byte byteValue( )	This Method return the value of the invoking object as a byte.
int compareTo(Float number)	Compare the numerical value of the invoking object with that of number. It will return 0 if the values are equal. It will return a negative value if the invoking object has a lower value. It will return a positive value if the invoking object has a greater value.
double doubleValue( )	This Method return the value of the invoking object as a double.
boolean equals(Object Obj)	This Method return true if the invoking float object is equivalent to Obj. Otherwise, it return false.
float floatValue( )	This Method return the value of the invoking object as a float.
int intValue( )	This Method return the value of the invoking object as an int.
boolean isInfinite ()	This Method return true if the invoking object has an infinite value.
static Boolean isInfinite(float number)	This Method return true if the number specifies an infinite value. Otherwise, it will return false.
Boolean isNan ()	This Method return true if the invoking object contains a value that is not a number. Otherwise, it will return false.
static Boolean isNaN(float number)	This Method return true if number specifies an infinite value. otherwise, it returns false.
long longValue( )	This Method return the value of the invoking object as a long.
static long parseByte(String s) throws NumberFormat Exception	This Method return the float equivalent of the number contained in the string specified by s. This method will used radix 10.
short shortValue( )	This Method return the value of the invoking object as a short.
static toString( )	This Method return a string that contains the decimal equivalent of the invoking object.

static String toString(float number)	This Method return a string that contains the decimal equivalent of number.
static Long valueOf(String s) throws NumberFormat Exception	This Method return a Short object that contains the value specified by the string in s.

The following program show the use of some methods defined in Float class :

```
class FloatDemo
{
 public static void main(String args[])
 {
  Float a1 = new Float(23.345);
  Float a2 = new Float(745.22f);
  Float a3 = new Float("789.35");


  System.out.println("\n value of a1 object = " + a1);
  System.out.println("\n value of a1 object = " + a2);
   System.out.println("\n byte value of Float object a1  = " +
a1.byteValue());
   System.out.println("\n Float value of Float object a3  = " +
a3.floatValue());
  System.out.println("\n  is a1 = a2 ? : " + a1.compareTo(a2));
  float a4 = Float.parseFloat("688.564");
  System.out.println("\n Float value of a4 =" + a4 );
 }
}
```



```
ankit@LP-14: ~/Desktop/BAOU_1/PORGRAMME/BLOCK_2
File Edit View Search Terminal Help
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac DemoIterator.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java DemoIterator
12
8
2
23
[12, 8, 23]
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$
```

Figure 7.4 Output of Program

❑ **Check Your Progress – 4 :**

1. Explain Float wrapper Class.

....

....

7.8 Double Class :

The Double class is a wrapper class for the double data type. The constructors for Double class are as follow :

❖ **Constructors :**

Double(double num);

Double(String str);

Here num is double type and str is string representation of a double.

The bellow table 7.7 gives a brief description of methods available under Double Class :

Table 7.7 : Methods defined in Double Class

Method	Description
byte byteValue( )	To return the value of the invoking object as a byte.
int compareTo(Double number)	To compare the numerical value of the invoking object with that of number. It will return 0 if the values are equal. It will return a negative value if the invoking object has a lower value. It will return a positive value if the invoking object has a greater value.
double doubleValue( )	To return the value of the invoking object as a double.
boolean equals(Object Obj)	To return true if the invoking double object is equivalent to Obj. Otherwise, it return false.
float floatValue( )	To return the value of the invoking object as a float.
int intValue( )	To return the value of the invoking object as an int.
boolean isInfinite ()	To return true if the invoking object has an infinite value.
static Boolean isInfinite(double number)	To return true if the number specifies an infinite value. Otherwise, it will return false.
Boolean isNan ()	To return true if the invoking object contains a value that is not a number. Otherwise, it will return false.
static Boolean isNaN(double number)	To return true if number specifies an infinite value. otherwise, it returns false.
long longValue( )	To return the value of the invoking object as a long.
static long parseByte(String s) throws NumberFormat Exception	To return the float equivalent of the number contained in the string specified by s. This method will used radix 10.

short shortValue( )	To return the value of the invoking object as a short.
static toString( )	To return a string that contains the decimal equivalent of the invoking object.
static String toString(double number)	To return a string that contains the decimal equivalent of number.
static Long valueOf(String s) throws NumberFormat Exception	To return a Short object that contains the value specified by the string in s.

7.9 Boolean Class :

The Boolean class is a wrapper class for the boolean data type. The constructors for Boolean class are as follow :

❖ **Constructors :**

Boolean (Boolean num);

Boolean (String str);

Here num is boolean type and str is string representation of a boolean.

The bellow table 7.8 gives a brief description of methods available under Boolean Class :

Table 7.8 : Methods defined in Boolean Class

Method	Description
Boolean BooleanValue ()	This method return boolean equivalent.
Boolean equals(Object obj)	This method return true if the invoking object is equivalent to obj. Otherwise this method will return false.
Static Boolean getBoolean (String propertyName)	This method return true if the system property specified by propertyName is true. Otherwise this method will return false.
int hashCode( )	This method return the hash code for the invoking object.
String toString ( )	This method return the string equivalent of the invoking object.
Static Boolean valueOf(Strign str)	This method return true if str contains the string "true". Otherwise this method will return false.

7.10 Character Class :

The Character class is a wrapper class for the char data type. The constructors for Character class are as follow :

❖ **Constructors :**

Character (char c);

Here c is char type.

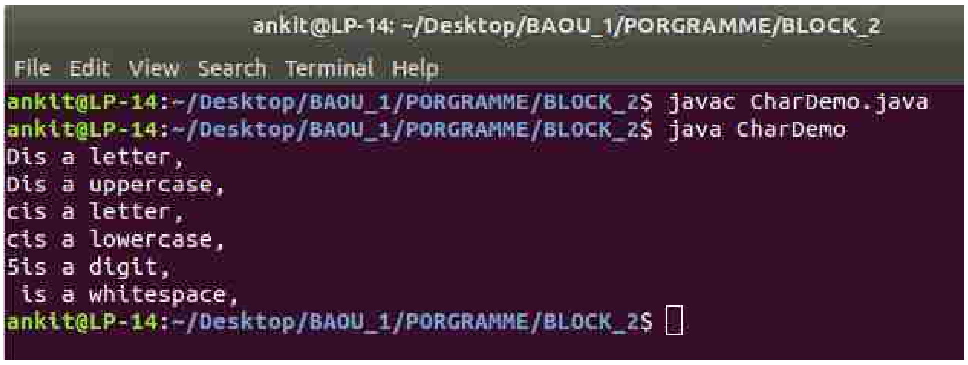
The bellow table 7.9 gives a brief description of methods available under Character Class :

Table 7.9 : Methods defined in Character Class

Method	Description
static Boolean isDefined(char c)	To return true if c is defined by Unicode. Otherwise, this method will return false.
static Boolean isDigit(char c)	To return true if c is a digit. Otherwise, this method will return false.
static Boolean isIdentifierIgnorable(char c) static Boolean isISoControl(char c)	To return true if c should be ignored in an identifier. Otherwise, this method will return false. To return true if c is an ISO control character. Otherwise, this method will return false.
static Boolean isJavaIdentifier Part(char c)	To return true if c is allowed as part of a Java identifier. Otherwise, this method will return false.
Static Boolean isJavaIdentifier Start(char c)	To return true if c is allowed as part of first character of a Java Identifier. Otherwise, this method will return false.
Static Boolean isLetter (char c)	To return true if c is a letter. Otherwise, this method will return false.
Static Boolean isLowerOrDigit (char c)	To return true if c is a letter of a digit. Otherwise, this method will return false.
Static Boolean isLowerCase (char c)	To return true if c is a lowercase letter. Otherwise, this method will return false.
Static Boolean isSpaceChar (char c)	To return true if c is Unicode space character. Otherwise, this method will return false.
Static Boolean isTitleCase(char c)	To return true if c is a Unicode titlecase character. Otherwise, this method will return false.
Static Boolean isUnicode IdentifierPart(char c)	To return true if c is allowed as part of a Unicode identifier. Otherwise, this method will return false.
Static Boolean isUnicode IdentifierStart(char c)	To return true if c is allowed as the first character of a Unicode identifier. Otherwise, this method will return false.
Static Boolean isUpperCase (char c)	To return true if c is an uppercase letter. Otherwise, this method will return false.
Static Boolean isWhitespace (char c)	To return true if c is whitespace. Otherwise, this method will return false.
Static Boolean toLowerCase (char c)	To return lowercase equivalent of c.
Static Boolean toTitleCase (char c)	To return titlecase equivalent of c.
Static char to UpperCase(char c)	To return uppercase equivalent of c.

The following program show the use of some methods defined in Character class :

```
class CharDemo
{
 public static void main (String args[])
 {
  char c[] = {'D', 'c', '@', '5', ' '};
  for(int i=0; i<c.length; i++)
  {
   if(Character.isDigit(c[i]))
    System.out.println(c[i]+ "is a digit,");
   if(Character.isLetter(c[i]))
    System.out.println(c[i]+ "is a letter,");
   if(Character.isWhitespace(c[i]))
    System.out.println(c[i]+ "is a whitespace,");
   if(Character.isUpperCase(c[i]))
    System.out.println(c[i]+ "is a uppercase,");
   if(Character.isLowerCase(c[i]))
    System.out.println(c[i]+ "is a lowercase,");
  }
 }
}
```


ankit@LP-14: ~/Desktop/BAOU_1/PORGRAMME/BLOCK_2
File Edit View Search Terminal Help

```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac CharDemo.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java CharDemo
Dis a letter,
Dis a uppercase,
cis a letter,
cis a lowercase,
5is a digit,
 is a whitespace,
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$
```

Figure 7.5 Output of Program

❑ **Check Your Progress – 5 :**

1. Explain Character wrapper Class.

....

....

7.11 String Class :

In Java, String is defined as a sequence of characters. Java provides two classes for handling String, String and StringBuffer. String class deals with strings that are not altered after creation. StringBuffer class deals with strings that need alteration after they are created.

The constructors for String class are as follow :

❖ **Constructors :**

```
String stringName;
stringName = new String("ABC");

String StringName = new String("ABC");

char[] str = {'j','a','v','a'};
String strone = new String(str);

String str = "java";
```

The bellow table 7.10 gives a brief description of methods available under String Class :

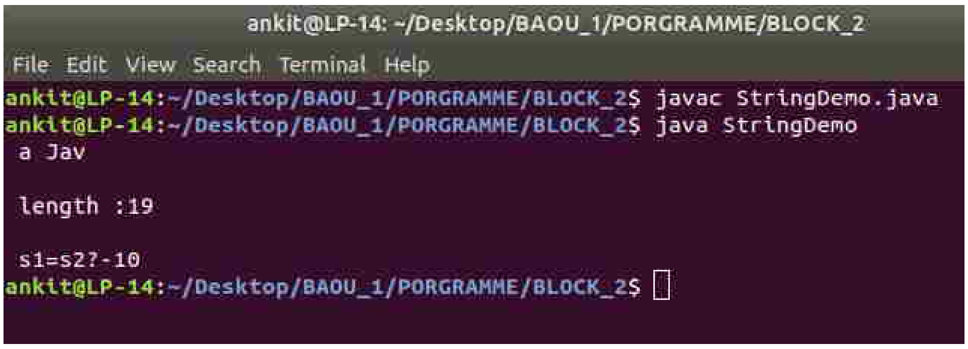
Table 7.10 : Methods defined in String Class

Method	Description
char charAt(int index)	This method returns the character at the index position of the invoking string object.
void getChars(int start, int end, char target[], int target start)	It copies characters from object string starting at 'start' up to 'end' character in to 'target', starting at 'target start'
byte[] getBytes()	This method returns an array of characters as bytes from the String object.
boolean equals(Object str)	This method returns true if str contains the same string as that in the invoking object.
boolean equalsIgnoreCase (String str)	This method returns true if str contains the same string as in the invoking object by ignoring the case
boolean regionMatches (int start, String s2, int s2startindex, int numchars)	This method compare a region of the invoking object based on parameters pass.
boolean endsWith(String str)	This method returns true if the invoking String object end withstr.
boolean startWith(String str)	This method returns true if the invoking String object starts with str
int compareTo(String str)	This method compare the invoking String object with str.
int indexOf(int ch)	This method returns the index of first occurrence of the character ch in the invoking string object
int lastindexOF(int ch)	This method returns the index of last occurrence of the character ch in the invoking string object
int indexOf(String str)	This method returns the index of first occurrence of the string str in the invoking string object

int lastindexOF(String str)	This method returns the index of last occurrence of the string str in the invoking string object
String substring(int startIndex)	This method return a substring starting at startIndex till the end of the invoking String object
String substring(int startIndex, int endIndex)	This method return a substring starting at startIndex up to the endindex of the invoking String object
String concat(String str)	This method returns a new String after appending the str to the invoking String object
String replace(char existing Char, char newChar)	This method returns the new string created by replacing existingChar with newChar
String trim()	This method returns a new String after removing the leading and trailing white spaces of the invoking String object
String toLowerCase()	This method converts the uppercase characters of invoking String object to lowercase
String toUpperCase()	This method converts the lowercase characters of invoking String object to uppercase
int length()	This method return the number of characters in the invoking String object

The following program show the use of some methods defined in String class :

```
class StringDemo
{
 public static void main (String args[ ])
 {
  String s1 = "This is a Java Text";
  String s2 = "This is a Text";
  char  data[] = new char [10];

  s1.getChars(7,13, data, 0);
  System.out.println(data);

  int count = s1.length();
  System.out.println("\n length :" + count);

  System.out.println("\n s1=s2?" + s1.compareTo(s2));
 }
}
```


ankit@LP-14: ~/Desktop/BAOU_1/PORGRAMME/BLOCK_2
File Edit View Search Terminal Help

```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac StringDemo.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java StringDemo
 a Jav

 length :19

 s1=s2?-10
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ 
```

Figure 7.6 Output of Program

❑ **Check Your Progress – 6 :**

1. Explain String Class.

....

....

2. Java provide ________ abstract class for the wrapper the data type.

(A) Integer (B) Number (C) Data Type (D) String

3. ________ return the byte value of the invoking object as a byte in Byte class.

(A) byteValue() (B) doubleValue()

(C) byte() (D) ToByte()

4. ________ method compare the numerical value of the invoking object with that of byte under Short class.

(A) compareTo(Short obj) (B) compareTo(int obj)

(C) compareTo(Float obj) (D) compareTo(Double obj)

5. ________ method return the value of the invoking object as a long under Integer class.

(A) Long () (B) Longvalue()

(C) IntegerValue() (D) longValue()

6. ________ method return the true if the invoking object contains a value that is not a number under Double class.

(A) isNan() (B) isnotnumber()

(C) isnumber() (D) isdigit()

7. The isInfinite() method return true if the invoking object has an infinite value under Float class.

(A) True (B) False

8. Boolean isDigit(char c) method return true if char c is a digit under Character class.

(A) True (B) False

9. ________ method return lowercase equivalent of character under Character class.

(A) toLowerCase(char c) (B) LowerCase(char c)

(C) toLowerCase() (D) toSmallCase(char c)

10. Java provides two classes for handling String, String and StringBuffer.

    (A) True (B) False

11. String class deals with strings that can altered after creation

    (A) True (B) False

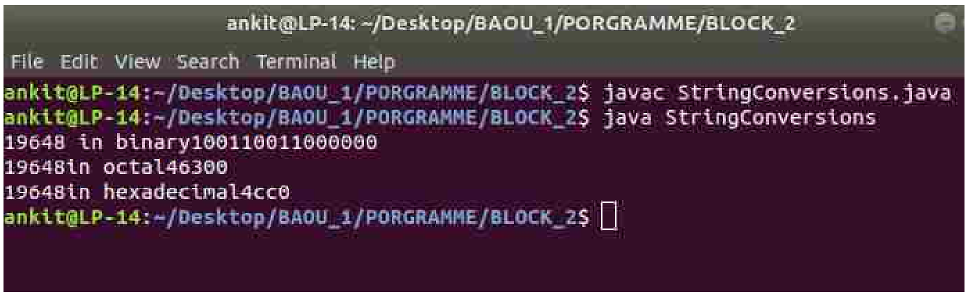
7.12 Converting Numbers to and from Strings :

Java provides an easy way to convert numbers into string. The Byte, Short, Integer and Long classes provide the parseByte( ), parseShort( ), parseInt( ) and parseLong( ) methods, respectively. These methods return the byte, short, int or long equivalent of the numeric string with which they are called.

The given below program demonstrates the use of parseInt( ). It finds the sum of a list of integers entered by the user. It reads the integers using readLine ( ) and uses parseInt( ) to convert these strings into their int equivalents.

The following program show the use convert an integer into binary, hexadecimal and octal :

```
class StringConversions
{
 public static void main (String args[])
 {
  int num=19648;
 System.out.println(num + " in binary" + Integer.toBinaryString(num));
  System.out.println(num + "in octal" + Integer.toOctalString (num));
 System.out.println(num + "in hexadecimal" + Integer.toHexString (num));
 }

}
```



```
ankit@LP-14: ~/Desktop/BAOU_1/PORGRAMME/BLOCK_2
File Edit View Search Terminal Help
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac StringConversions.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java StringConversions
19648 in binary100110011000000
19648in octal46300
19648in hexadecimal4cc0
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ 
```

Figure 7.7 Output of Program

7.13 Let Us Sum Up :

In this Unit we have learned about a superclass which is defined by the abstract class Number that implements the classes that wrap the numeric type's byte, short, int, long, float and double. Number possesses abstract methods that return the value of the object in each of the different number formats. That is, doubleValue ( ) returns the value as a double, floatValue ( ) returns the value as a float and so on. Double and Float are wrappers for floating–point values of type double and float respectively. The constructors of float are Float (double num) ,Float (float num) and Float (String str) throws

NumberFormatException. The Float objects can be constructed with values of type float or double. They can also be constructed from the string representation of a floating–point number. Whereas, the constructors for Double are shown as Double (double num), Double (String str) throws Number Format Exception. Double objects can be constructed with a double value or a string containing a floating–point value.

Java provides an easy way to convert numbers into string. The Byte, Short, Integer and Long classes provide the parseByte( ), parseShort( ), parseInt ( ) and parseLong( ) methods, respectively. These methods return the byte, short, int or long equivalent of the numeric string with which they are called.

7.14 Suggested Answer for Check Your Progress :

❑ **Check Your Progress 1 :**

See Section 7.3

❑ **Check Your Progress 2 :**

See Section 7.4

❑ **Check Your Progress 3 :**

See Section 7.5

❑ **Check Your Progress 7 :**

See Section 7.7

❑ **Check Your Progress 5 :**

See Section 7.10

❑ **Check Your Progress 6 :**

1 : See Section 7.11	**2 :** B	**3 :** A	**4 :** A	**5 :** D
6 : A	**7 :** A	**8 :** A	**9 :** A	**10 :** A
11 : B				

7.15 Glossary :

1. **Number Class** – Java provide Number class with six subclasses. Using the subclasses we can create the object of basic data type.
2. **Byte Class** – The Byte class is a wrapper class for the byte data type.
3. **Short Class** – The Short class is a wrapper class for the short data type.
4. **Integer Class** – The Integer class is a wrapper class for the int data type.
5. **Long Class** – The Long class is a wrapper class for the long data type.
6. **Float Class** – The Float class is a wrapper class for the float data type.
7. **Double Class** – The Double class is a wrapper class for the double data type.
8. **Boolean Class** – The Boolean class is a wrapper class for the boolean data type.
9. **Character Class** – The Character class is a wrapper class for the char data type.
10. **String Class** – String is defined as a sequence of characters. Java provides two classes for handling String, String and StringBuffer.

7.16 Assignment :

1. Write a note on Number Class.
2. Write a note on Wrapper Class.

7.17 Activities :

1. Write a program to show the use of String Class.
2. Write a program to show the use of Double Class.
3. Write a program to show the use of Long Class.
4. Write a program to show the use of Boolean Class.

7.18 Case Study :

1. Prepare the Chart of five method from each wrapper class.

7.19 Further Reading :

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000.
2. Java 2, the Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999.
3. Programming with Java, Ed. 2, E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000.
4. The Java Tutorial, Ed. 2, 2 volumes, Mary Campione and Kathy Walrath, Addison Wesley Longmans, 1998.
5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele & Gilad Bracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000.
6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition, 2000

Unit 08

JAVA COLLECTION FRAMEWORK

UNIT STRUCTURE

8.0 Learning Objectives :

After learning this unit, you will be able to understand the use of :

- Implementation of Data Structure concept using Java Collection Classes
- Collection interface
- List interface
- LinkedList Class
- Array List Class
- Stack Class
- Queue Interface
- TreeSet Class
- HasSet Class
- TreeMap Class
- HasMap Class

- Iterator
- List Iterator

8.1 Introduction :

The Java platform includes a collections framework. A collection is an object that represents a group of objects. The collection framework provide a well–designed set of interface and classes for storing and manipulating of data as a single unit. Java Collections can achieve all the operations that we perform on a data such as searching, sorting, insertion, manipulation, and deletion.

The collections framework was designed to meet several goals, such as :

- Reduces programming effort by providing data structures and algorithms so you don't have to write them yourself.
- Increases performance by providing high–performance implementations of data structures and algorithms.
- Provides interoperability between unrelated APIs by establishing a common language to pass collections back and forth.
- Reduces the effort required to learn APIs by requiring you to learn multiple ad hoc collection APIs.
- Reduces the effort required to design and implement APIs by not requiring you to produce ad hoc collections APIs.
- Fosters software reuse by providing a standard interface for collections and algorithms with which to manipulate them.

The entire collections framework is designed around a set of standard interfaces. A collections framework is a unified architecture for representing and manipulating collections. In addition to collections, the framework defines several map interfaces and classes. Maps store key/value pairs. Although maps are not collections in the proper use of the term, but they are fully integrated with collections.

8.2 Collection Interface :

Bellow diagram show the hierarchy of Collection framework. The Java.util package contains all the classes and interfaces for the collection framework.

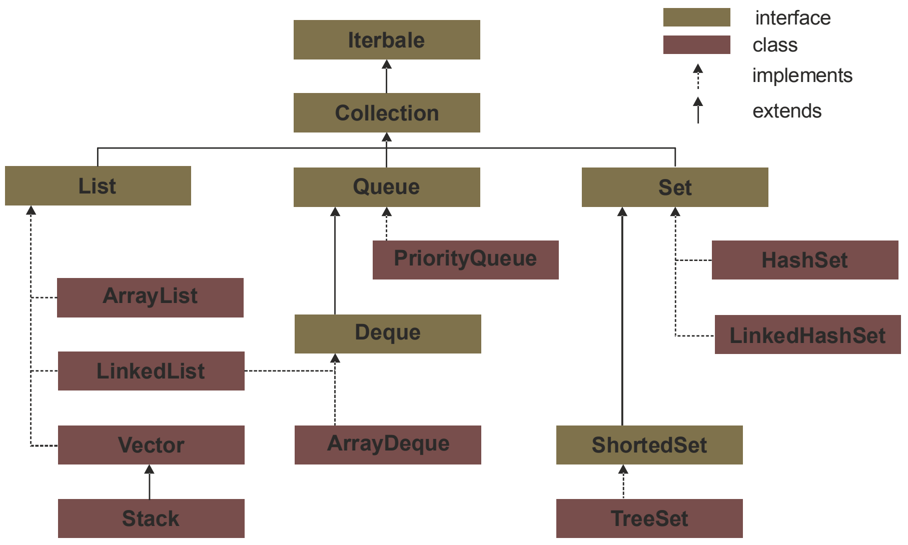


Figure 8.1 Collection Interface

The parent interface of Collection interface is Iterable interface. We also see that child interface of Collection interface are List, Queue and Set interfaces. In this unit we study the interfaces and classes available under the Collection interface.

The Following points need to be remembered regarding Collection Framework :

- The Collection interface is a group of objects, with duplicates allowed.
- The List interface extends Collection, allows duplicates and introduces positional indexing.
- The Set interface extends Collection but forbids duplicates
- The Collection interface is used to represent any group of objects or elements.
- This interface is implemented by all collection classes.
- The interface supports basic operations like adding and removing.

The bellow table 8.1 gives a brief description of methods available under Collection Interface :

Table 8.1 : Methods defined in Collection Interface

Method	Description
boolean add(Object obj)	This method add obj to the invoking collection.
boolean add(Collection c)	Add all the elements of c to the invoking collection.
Void clear()	Removes all elements from the invoking collection.
boolean contains(Object obj)	Returns true if obj is an element of the invoking collection.
boolean containsAll (collection c)	Returns true if the invoking collection contains all elements of c.
boolean equals(Object obj)	Returns true if the invoking collection is equals
boolean isEmpty()	Returns true if the invoking collection is empty.
Iterator iterator()	Returns an iterator for the invoking collection.
boolean remove(Object obj)	Remove one instance of obj from the invoking collection.
boolean removeAll (Collection c)	Remove all elements of c from the invoking collection.
boolean retainAll (Collection c)	Remove all elements from the invoking collection except those in c.
Int size()	Returns the number of elements held in the invoking collection.
Object [] toArray()	Returns an array that contains all the elements storred in the invoking collection.
Object [] toArray (Objectarray [])	Returns an array containing only those collection elements whose type matches that of the array.

8.3 List Interface :

The Collection interface extends List interface. The List interface has four concrete subclasses, LinkList, ArrayList, Vector and Stack classes. The List interface provides the following facilities :

- Permitting duplicates.
- The interface adds position oriented operations.
- The first element in the list starts at index 0.
- Elements can be added and accessed by their position in this list.

The bellow table 8.2 gives a brief description of methods available under List Interface :

Table 8.2 : Methods defined in List Interface

Method	Description
void add(index, object obj)	Insert into the invoking list at the index passedin index.
boolean addAll(int index, Collection c)	Inserts all elements of c into the invoking list at the index passed in index.
object get(int index)	Returns the object stored at the speccified index within the invoking collection.
int indexOf(object obj)	Returns the index of the first instance of obj in the invoking list. Return – 1 if obje is not an element.
int lastIndexOf(Object obj)	Return the index of the last instance of obj in the invoking list.Return -1 if obje is not an element.
listIterator listIterator(int index)	Return an iterator to the invoking list that begins at the specified index.
object remove(int index)	Removes the element at position index from the invoking list and returns the deleted elements.
object set(int index, Object obj)	Assigns obj to the location specified by index within the invoking list
list subList(int start, int end)	Returns a list that includes elements from start and end.

8.4 LinkedList Class :

Linked list is a fundamental data structure that contains records. LinkedList implements the Collection interface. A record contains data as well as a reference to the next record. A record can be inserted or removed at any point in the Linked List. Only sequential access is allowed.

It uses a doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order. In LinkedList, the manipulation is fast because no shifting is required. The constructors for short class are as follow :

❖ **Constructors :**

LinkedList()

LinkedList(Collection c)

Here C is Collection object.

The bellow table 8.3 gives a brief description of methods available under LinkedList class :

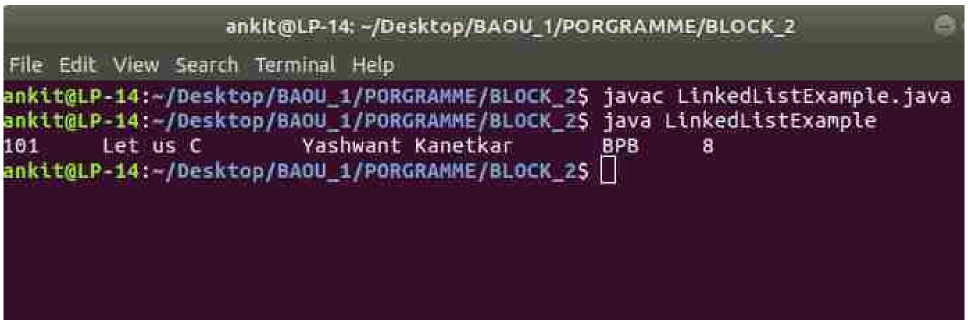
Table 8.3 : Methods defined in LinkedList Class

Method	Description
void add(int index, Object element)	It is used to insert the specified element at the specified position index in a list.
void addFirst(Object o)	It is used to insert the given element at the beginning of a list.
void addLast(Object o)	It is used to append the given element to the end of a list.
int size()	It is used to return the number of elements in a list
boolean add(Object o)	It is used to append the specified element to the end of a list.
boolean contains(Object o)	It is used to return true if the list contains a specified element.
boolean remove(Object o)	It is used to remove the first occurence of the specified element in a list.
Object getFirst()	It is used to return the first element in a list.
Object getLast()	It is used to return the last element in a list.
int indexOf(Object o)	It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.
int lastIndexOf(Object o)	It is used to return the index in a list of the last occurrence of the specified element, or -1 if the list does not contain any element.

The following program show the use of some methods defined in LinkedList class :

```
import java.util.*;

class Book
{
 int id;
 String name,author,publisher;
 int quantity;

 public Book(int id, String name, String author, String publisher, int
quantity)
```

```
 {
        this.id = id;
        this.name = name;
        this.author = author;
        this.publisher = publisher;
        this.quantity = quantity;
 }
}

public class LinkedListExample
{
 public static void main(String[] args)
 {
        //Creating list of Books
        List<Book> list=new LinkedList<Book>();

   //Creating Books
      Book b=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);

   //Adding Books to list
        list.add(b);

     System.out.println(b.id+"\t"+b.name+"\t"+b.author+"\t"+b.publisher+
"\t"+b.quantity);
       }
}
```



```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac LinkedListExample.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java LinkedListExample
101     Let us C        Yashwant Kanetkar       BPB     8
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ 
```

ankit@LP-14: ~/Desktop/BAOU_1/PORGRAMME/BLOCK_2
File Edit View Search Terminal Help

Figure 8.2 Output of Program

8.5 ArrayList Class :

The ArrayList class implements the List interface. It uses a dynamic array to store element of different data types. The ArrayList class maintains the insertion order. The elements stored in the ArrayList class can be randomly accessed. It inherits Abstract List class and implements List interface. Java ArrayList class can contain duplicate elements. Java ArrayList class maintains insertion order.Java ArrayList allows random access because array works at the index basis. In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list. The constructors for short class are as follow :

❖ **Constructors :**

ArrayList()

ArrayList(Collection c)

ArrayList(int capacity)

Here C is Collection object and capacity represent the no of element hold by ArrayList class.

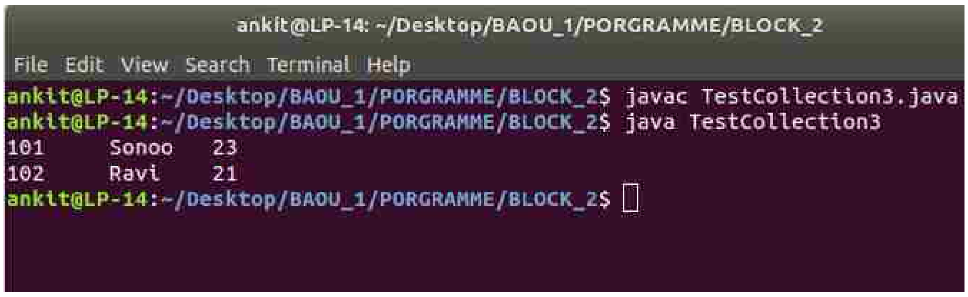
The bellow table 8.4 gives a brief description of methods available under ArrayList class :

Table 8.4 : Methods defined in ArrayList Class

Method	**Description**
void add(int index, Object element)	It is used to insert the specified element at the specified position index in a list.
boolean addAll(Collection c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
void clear()	It is used to remove all of the elements from this list.
int lastIndexOf(Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
Object[] toArray(Object[] a)	It is used to return an array containing all of the elements in this list in the correct order.
boolean add(Object o)	It is used to append the specified element to the end of a list.
boolean addAll(int index, Collection c)	It is used to insert all of the elements in the specified collection into this list, starting at the specified position.
Object clone()	It is used to return a shallow copy of an ArrayList.
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
void trimToSize()	It is used to trim the capacity of this ArrayList instance to be the list's current size.

The following program shows how to create ArrayList and add value in it.

```
//133

import java.util.*;

class Student
{
 int rollno;
 String name;
 int age;

 Student(int rollno,String name,int age)
 {
  this.rollno=rollno;
  this.name=name;
  this.age=age;
 }
}

public class TestCollection3
{
 public static void main(String args[])
 {
  //Creating user-defined class objects
  Student s1=new Student(101,"Sonoo",23);
  Student s2=new Student(102,"Ravi",21);

  //creating arraylist
  ArrayList<Student> al=new ArrayList<Student>();
  al.add(s1);//adding Student class object
    al.add(s2);

  for(Student st:al)
  {
        System.out.println(st.rollno+"\t"+st.name+"\t"+st.age);
    }
  }
}
```


ankit@LP-14: ~/Desktop/BAOU_1/PORGRAMME/BLOCK_2
File Edit View Search Terminal Help

```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac TestCollection3.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java TestCollection3
101     Sonoo   23
102     Ravi    21
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$
```

Figure 8.3 Output of Program

❑ **Check Your Progress – 1 :**

1. Explain Byte ArrayList Class.
2. Explain the LinkedList Class

....

....

8.6 Stack Class :

Java Collection framework provides a Stack class which models and implements Stack data Structure. The stack is the subclass of Vector. It implements the last–in–first–out data structure, i.e., Stack. The class is based on the basic principle of last–in–first–out (LIFO). The class provides basic operation push and pop. The stack contains all of the methods of Vector class. The constructors for short class are as follow :

❖ **Constructors :**

Stack()

The insertion (push) and deletion (Pop) operation show in the bellow figure.

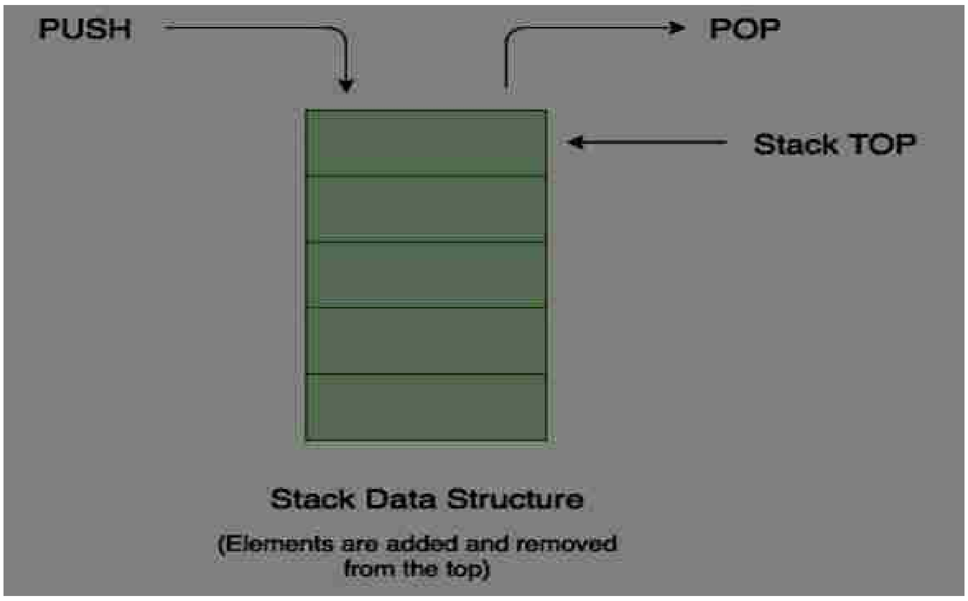


Figure 8.4 Stack Data Structure

The bellow table 8.5 gives a brief description of methods available under Stack class :

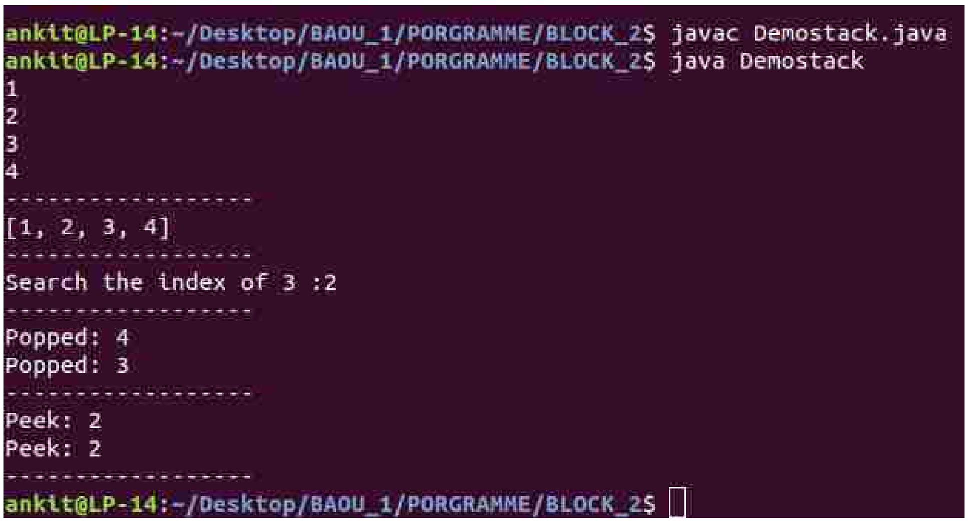
Table 8.5 : Methods defined in Stack Class

Method	Description
Object push(object element)	Pushes an element on the top of stack
Object pop()	Removes and returns the top element of the stack. An 'EmptyStackException' is thrown if we call pop() when the invoking stack is empty.
Object peek()	Returns the element on the top of the stack, but does not remove it.
Boolean empty()	It returns true if nothing is on the top of the stack. Else rturn fase.
Int search(object element)	It determines whether an object exists in the stack. If the elemetn is found, it returns the positions of the element from the top of the stack else return -1

The following program show the use of some methods defined in Stack class :

```
//134_35

import java.util.*;

class Demostack
{
 public static void main(String args[])
 {

  Stack<Integer> l2=new Stack<Integer>();
  l2.push(new Integer(1));
  l2.push(new Integer(2));
  l2.push(new Integer(3));
  l2.push(new Integer(4));

   for(int a:l2)
    System.out.println(a);

   System.out.println("------------------");
   System.out.println(l2);
   System.out.println("------------------");
   System.out.println("Search the index of 3 :" + l2.search(3));
   System.out.println("------------------");
```

```
        System.out.println("Popped:  "+l2.pop());
        System.out.println("Popped:  "+l2.pop());

        System.out.println("------------------");
        System.out.println("Peek:  "+l2.peek());
        System.out.println("Peek:  "+l2.peek());
        System.out.println("------------------");

    }
  }
```



```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac Demostack.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java Demostack
1
2
3
4
------------------
[1, 2, 3, 4]
------------------
Search the index of 3 :2
------------------
Popped: 4
Popped: 3
------------------
Peek: 2
Peek: 2
------------------
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$
```

Figure 8.5 Output of Program

8.7 Queue Interface :

The Queue interface present in the java.util package and extend the Collection interface. The class is based on the basic principle of last–in–first–out (FIFO). It is an ordered list of objects with its use limited to insert elements at the end of the list and deleting elements from the start of the list. Being an interface the queue needs a concrete classes for the declaration, that are, PriorityQueue, LinkedList, PriorityBlockingQueue.

The insertion and deletion operation show in the bellow figure.

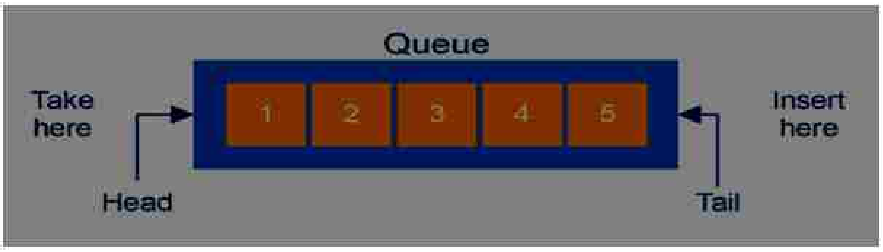


Figure 8.6 Queue Data Structure

The bellow table 8.6 gives a brief description of methods available under Queue interface :

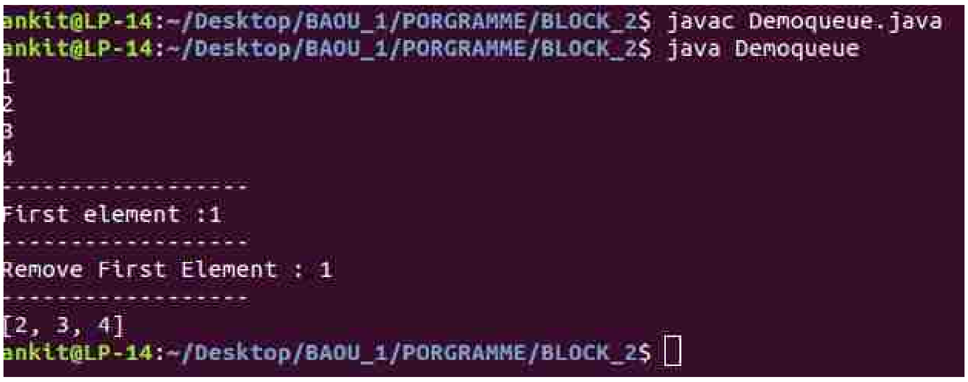
Table 8.6 : Methods defined in Queue interface

Method	Description
Boolean add(object)	It is used to insert the specified element into queue and return true upon success.
Boolean offer(object)	It is used to insert the specified element into this queue.
Objet remove()	It is used to retrieves and removes the head of this queue.
Object poll()	It is used to retrieves and removes the head of this queue, or returns null if this queue is empty.
Object element()	It is used to retrieves, but does not remove, the head of the queue.
Object peek()	It is used to retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

The following program show the use of some methods defined in Queue interface :

```
import java.util.*;

class Demoqueue
{
 public static void main(String args[])
 {

   Queue<Long> l3=new LinkedList<Long>();
   l3.add(new Long(1));
   l3.add(new Long(2));
   l3.add(new Long(3));
   l3.add(new Long(4));

    for(Long b:l3)
     System.out.println(b);
   System.out.println("------------------");

   //to display element at first position
   System.out.println("First element :" + l3.peek());
   System.out.println("------------------");
```

```
        //to remove element from first position
        System.out.println("Remove First Element : " + l3.poll());
        System.out.println("-----------------");
        System.out.println(l3);
        }
    }
```



```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac Demoqueue.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java Demoqueue
1
2
3
4
-----------------
First element :1
-----------------
Remove First Element : 1
-----------------
[2, 3, 4]
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$
```

Figure 8.7 Output of Program

❑ **Check Your Progress – 2 :**

1. Explain Byte Stack Class.
2. Explain the Queue Class

....

....

8.8 Set Interface :

A Set is an interface under Collection interface that cannot contain duplicate elements. It models the mathematical set abstraction. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited. We can store at most one null value in Set. The concept of union, intersection, and the differenece of a set are available in the set interface and supported by its subclasses. Set is implemented by HashSet, LinkedHashSet, and TreeSet classes.

The bellow table 8.7 gives a brief description of methods available under Set interface :

Table 8.7 : Methods defined in Set interface

Method	Description
add( )	Adds an object to the collection.
clear( )	Removes all objects from the collection.
contains( )	Returns true if a specified object is an element within the collection.
isEmpty( )	Returns true if the collection has no elements.
iterator( )	Returns an Iterator object for the collection, which may be used to retrieve an object.
remove( )	Removes a specified object from the collection.
size( )	Returns the number of elements in the collection.

8.9 TreeSet Class :

The TreeSet class implements the Set and SortedSet interface. It uses the tree to storage of its element. It useful when one needs to extract elements from a collection in a sorted manner. TreeSet offers a strict control over the order of elements in the collection. The collection is a sorted collection. It may not offer you the best performance in terms of retrieving elements speedily. Does not permit null in the collection. Java TreeSet class is non synchronized. Java TreeSet class maintains ascending order.

The constructors for short class are as follow :

❖ **Constructors :**

Public TreeSet()

Public TreeSet(Collection C)

Public TreeSet(Comparator C)

Public TreeSet(SortedSet S)

The bellow table 8.8 gives a brief description of methods available under Stack class :

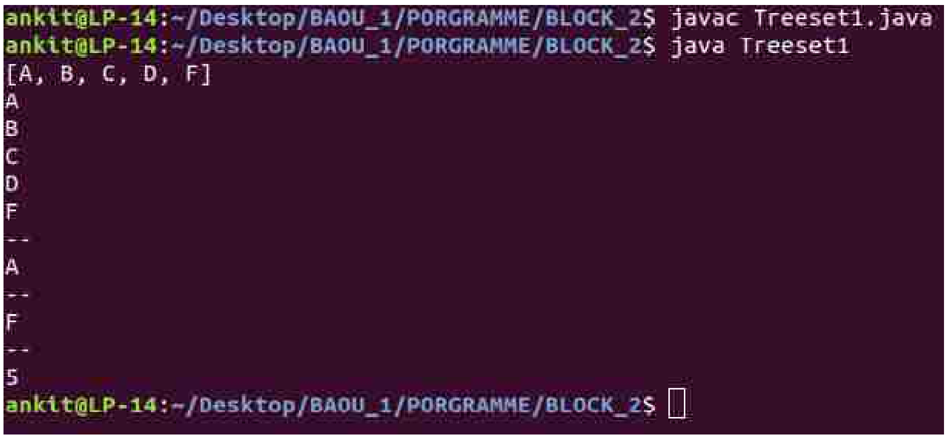
Table 8.8 : Methods defined in TreeSet Class

Method	Description
Comparator comparator()	Returns the comparator used to order this sorted set, or null if this tree set uses its elements naturl ordering
Object first()	Returns the first element currently in ths sorted set
Object last()	Return the last element currently in the sorted set
SortedSettailSet(Order fromElement)	Return a view of the portion of this set whose elements is greater than or equal to fromElement. The returned sorted set.
SortedSet headset(Object fromElement)	Return a view of the portion of this set this set whose elements are strictly less than toElement. Te returned sorted set in backed by this set.

The following program show the use of some methods defined in TreeSet class :

```
//138

import java.util.*;

class Treeset1
{
 public static void main(String args[])
 {
  TreeSet<String> t1=new TreeSet<String>();
```

```
        t1.add("D");
        t1.add("C");
        t1.add("B");
        t1.add("D");
        t1.add("D");
        t1.add("A");
        t1.add("F");

        System.out.println(t1);

        for(String s:t1)
         System.out.println(s);//iterating

        System.out.println("--");
        System.out.println(t1.first());
        System.out.println("--");
        System.out.println(t1.last());
        System.out.println("--");
        System.out.println(t1.size());
      }
    }
```



```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac Treeset1.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java Treeset1
[A, B, C, D, F]
A
B
C
D
F
--
A
--
F
--
5
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ 
```

Figure 8.8 Output of Program

8.10 Hashset Class :

HashSet class implements the Set interface. It does not guarantee that the order will remain constant over time. This class permits the null element. It used for storing the duplicate– free collection. For effectively storing and retrieving the elements but the order is not guaranteed by this class. To retrieve the elements in a sorted order. It allows null values. The constructors for short class are as follow :

❖ **Constructors :**

Public HashSet()

Public HashSet(Collection C)

Public HashSet(int initialCapacity)

❑ **Check Your Progress – 3 :**

1. Explain Set interface with subclasses.

....

....

8.11 Map Interface :

Map is just collection of Pairs. The interfaces Map and Collection are distinct. A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys. A Map is useful if you have to search, update or delete elements on the basis of a key.

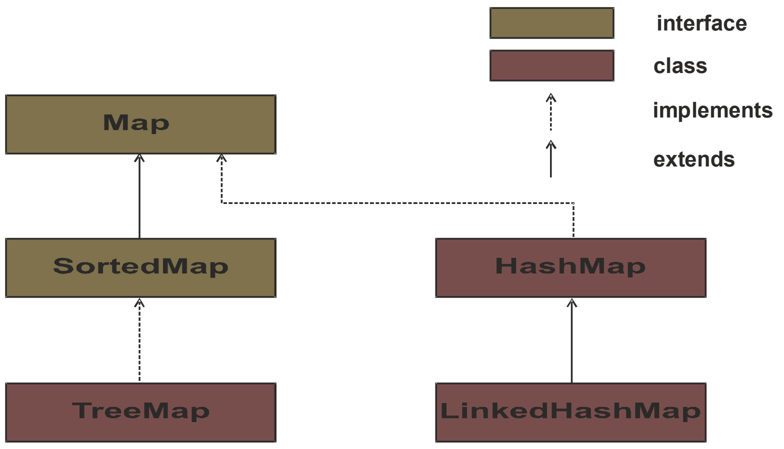


Figure 8.9 Map Interface

Map Interface has SortedMap interface and following child class :

- HasMap Class – HashMap is the implementation of Map, but it doesn't maintain any order.
- LinkedHasMap Class – LinkedHashMap is the implementation of Map. It inherits HashMap class. It maintains insertion order.
- TreeMap Class – TreeMap is the implementation of Map and SortedMap. It maintains ascending order.

The bellow table 8.9 gives a brief description of methods available under Map interface :

Table 8.9 : Methods defined in Map interface

Method	Description
Void clear()	Remove all key value pairs from the invoking map
Boolean containsKey (Object k)	Returns "true" if the invoking map contains k as a key.
Boolean continsValue (Object v)	Returns "true" if the invoking map contains v as a value.

Set entrySet()	Return a set that contains the entries in the map. The set contains objects of type Map.Entry
Boolean equals(object obj)	Returns "true" if obj is a Map and contains the same entries
Object get(object k)	Returns the value associated with the key k.
Int hashCode()	Returns the hash code for the invoking map.
Boolean isEmpty()	Returns "true" if the invoing map is empty.
Set KeySet()	Returns a Set that contains the keys in the invoking map.
Object put(object k, object v)	Puts an entry in the invoking map, overwritten any revious value associated with the key. The key and value are k and v respectively
Object get(object k)	Returns the value associated with the key k.
Int hashCode()	Returns the hash code for the invoking map.
Boolean isEmpty()	Returns "true" if the invoing map is empty.
Set KeySet()	Returns a Set that contains the keys in the invoking map.
Object put(object k, object v)	Puts an entry in the invoking map, overwritten any revious value associated with the key. The key and value are k and v respectively

8.12 TreeMap Class :

TreeMap is implemented from SortedMap. This class guarantees that the map will be in ascending key order, sorted according to the natural order for the key's class. TreeMap contains sorted mapping of key/value pairs. TreeMap Not allow null key and null value pairs to be stored. The constructors for short class are as follow :

❖ **Constructors :**

Public TreeMap()

Public TreeMap(Comparator c)

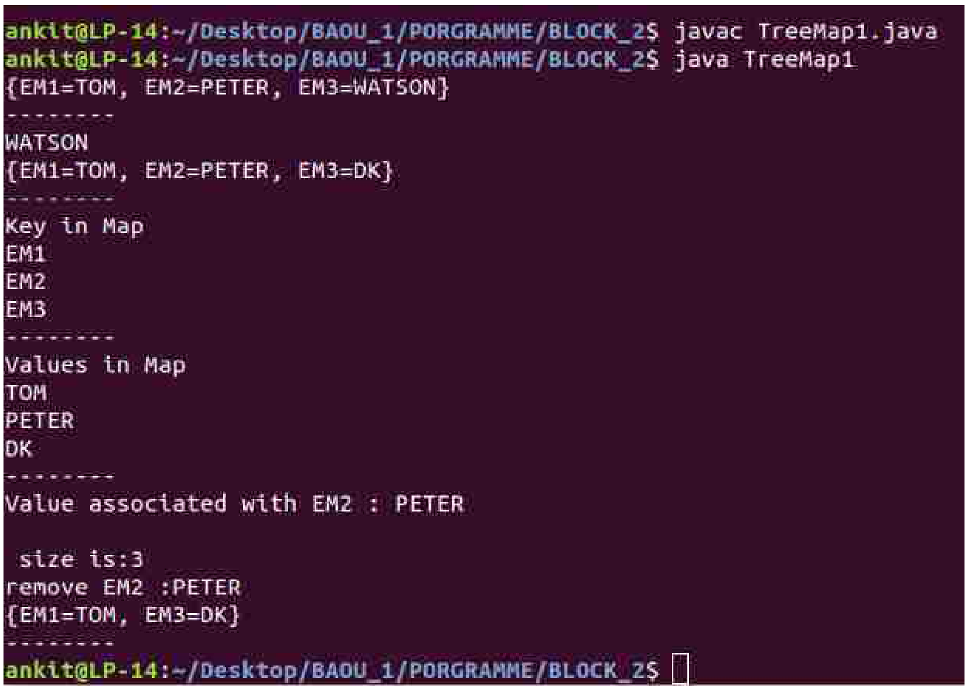
Public TreeMap(Map p)

Public TreeMap(SortedMap m)

The following program show the use of some methods defined in TreeMap class :

```
import java.util.*;

public class TreeMap1
{
##9;public static void main(String args[])
##9;{
```

```
##9;##9;TreeMap<String, String> tmap = new TreeMap<String, String>();
##9;##9;tmap.put("EM1","TOM");
##9;##9;tmap.put("EM3","WATSON");
##9;##9;tmap.put("EM2","PETER");
##9;##9;
##9;##9;System.out.println(tmap);
##9;##9;System.out.println("--------");##9;##9;

##9;##9;System.out.println(tmap.put("EM3","DK"));//modify value
##9;##9;System.out.println(tmap);
##9;##9;System.out.println("--------");##9;

##9;##9;System.out.println("Key in Map");

##9;##9;for(String s:tmap.keySet())
##9;##9;##9;System.out.println(s);
##9;##9;System.out.println("--------");##9;



##9;##9;System.out.println("Values in Map");
##9;##9;
##9;##9;for(String s:tmap.values())
##9;##9;##9;System.out.println(s);
##9;##9;System.out.println("--------");##9;
##9;##9;
##9;##9;System.out.println("Value associated with EM2 :
"+tmap.get("EM2"));
##9;##9;System.out.println("\n size is:"+tmap.size());
##9;##9;System.out.println("remove EM2 :"+tmap.remove("EM2"));
##9;##9;System.out.println(tmap);
##9;##9;System.out.println("--------");##9;
##9;}
}
```



```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac TreeMap1.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java TreeMap1
{EM1=TOM, EM2=PETER, EM3=WATSON}
--------
WATSON
{EM1=TOM, EM2=PETER, EM3=DK}
--------
Key in Map
EM1
EM2
EM3
--------
Values in Map
TOM
PETER
DK
--------
Value associated with EM2 : PETER

 size is:3
remove EM2 :PETER
{EM1=TOM, EM3=DK}
--------
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ 
```

Figure 8.10 Output of Program

8.13 HasMap Class :

The HasMap class uses hashing as a technique to store key/value pairs so that the values can be searched efficiently according to the key. There order is not guaraanteed by HashMap. HashMap allow null key and null value pairs to be stored. It is not an ordered collection which means it does not return the keys and values in the same order in which they have been inserted into the HashMap. The constructors for short class are as follow :

❖ **Constructors :**

Public HasMap()

Public HasMap(Map m)

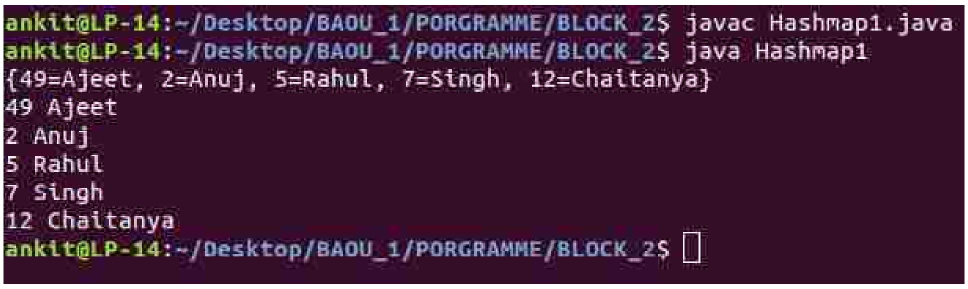
Public HasMap(int initialCapacity)

Public HasMap(int initialCapacity, float loadFactor)

The following program show the use of some methods defined in TreeMap class :

```
import java.util.*;


public class Hashmap1
{
 public static void main(String args[])
 {
  HashMap<Integer, String> hmap = new HashMap<Integer, String>();


  /*Adding elements to HashMap*/


  hmap.put(12, "Chaitanya");
```

```
          hmap.put(2, "Rahul");
            hmap.put(7, "Singh");
          hmap.put(49, "Ajeet");
            hmap.put(2, "Anuj");
     hmap.put(5, "Rahul");
     System.out.println(hmap);

     for(Map.Entry m:hmap.entrySet())
     {
         System.out.println(m.getKey()+" "+m.getValue());
       }
    }
   }
```



```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac Hashmap1.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java Hashmap1
{49=Ajeet, 2=Anuj, 5=Rahul, 7=Singh, 12=Chaitanya}
49 Ajeet
2 Anuj
5 Rahul
7 Singh
12 Chaitanya
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$
```

Figure 8.11 Output of Program

❑ **Check Your Progress – 4 :**

1. Explain Byte TreeMap Class.
2. Explain the HasMap Class

....

....

3. The collection interface is availabel under ________ package.

   (A) Java.IO (B) java.net (C) Java.lang (D) Java.util

4. The Vector class is derived from ________ interface.

   (A) Set (B) Queue (C) Stack (D) List

5. The ________ class's record contains data as well as a reference to the next record.

   (A) LinkedList (B) ArrayList (C) Vector (D) Stack

6. The Push and Pop operation available under ______ class.

   (A) List (B) HasMap (C) ArrayList (D) Stack

7. The ________ collection is a sorted collection.

   (A) TreeSet (B) ArrayList (C) Map (D) LinkedList

8. A ________ contains values on the basis of key.

   (A) Map (B) List (C) Queue (D) Set

8. The Iterator allows us to traverse the collection, access the data element.

   (A) True (B) False

9. The parent interface of Collection interface is Iterable interface.

   (A) True (B) False

10. The List interface Permitting duplicates value.

    (A) True (B) False

11. The ArrayList class uses a dynamic array to store element of different data types.

    (A) True (B) False

8.14 Iterator :

The 'Iterator' is an interface which belongs to collection framework. It allows us to traverse the collection, access the data element and remove the data elements of the collection. we can traverse a List or Set in forward direction. Before you can access a collection through an iterator, you must obtain one. Each of the collection classes provides an iterator( ) method that returns an iterator to the start of the collection.

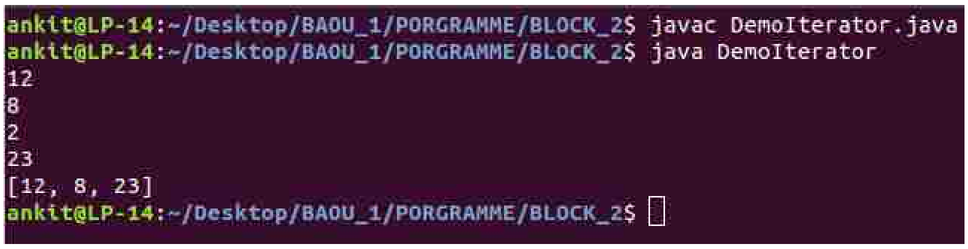
The bellow table 8.10 gives a brief description of methods available under Map interface :

Table 8.10 : Methods defined in Map interface

Method	Description
boolean hasNext( )	Returns true if there are more elements. Otherwise, returns false.
Object next( )	Returns the next element. Throws NoSuch ElementException if there is not a next element.
void remove( )	Removes the current element. Throws Illegal StateException if an attempt is made to call remove( ) that is not preceded by a call to next( ).

The following program show the use of some methods defined in Iterator :

```
import java.util.*;

public class DemoIterator
{
   public static void main(String[] args)
  {
       ArrayList<Integer> numbers = new ArrayList<Integer>();
       numbers.add(12);
       numbers.add(8);
       numbers.add(2);
       numbers.add(23);
Iterator<Integer> it = numbers.iterator();
    Iterator<Integer> it1 = numbers.iterator();
 // Print the item
while(it.hasNext())
{
 System.out.println(it.next());
}
// Remove the item

while(it1.hasNext())
{
       Integer i = it1.next();
       if(i < 5)
 {
         it1.remove();
       }
    }
    System.out.println(numbers);
 }
}
```



```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ javac DemoIterator.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$ java DemoIterator
12
8
2
23
[12, 8, 23]
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_2$
```

Figure 8.12 Output of Program

8.15 Let Us Sum Up :

In this unit we have learned various classes and interfaces available under the Collection framework. The collection interface derived from the Iterable interface. The collection framework available under the java.util package. The Collection interface has three child interfaces named List interface, Queue interface and Set interface. All the three interfaces has child classes or interface, used for the implementing Data structure concept for the group of data. Each child class has unique functionalities. The List interface allowed duplicated values to store. It works on position based stared with 0 index. Any element can be added or deleted based on position. The List interface has implemented by ArrayList class, LinkedList class, Vector class and Stack class. The Queue interface work on the principal of Fist–in–First–out (FIFO). The Queue interface has implemented by Priority Queue class, LinkedList class and PriorityBlockingQueue class. The Set interface does not allowed duplicates values to store. The Set interface has implemented by HasSet class, TreeSet and LindedHasSet class.

The Map interface is parallel to collection interface. It works on the principal of key and value. It contains values on the basis of key. The Map interface has implemented by HasMap class, LinkdedHasMap class and TreeMap class. The Iterator is available under the Collection framework. The Iterator allow us to traverse the collection, access the data element and remove the data element from the collection.

8.16 Suggested Answer for Check Your Progress :

- **Check Your Progress 1 :**

  See Section 8.4 & 8.5

- **Check Your Progress 2 :**

  See Section 8.6 & 8.7

- **Check Your Progress 3 :**

  See Section 8.8, 8.9, 8.10

- **Check Your Progress 4 :**

1 : See Section 8.12		**2 :** See Section 8.13		**3 :** D
4 : D	**5 :** A	**6 :** D	**7 :** A	**8 :** A
9 : A	**10 :** A	**11 :** A	**12 :** A	

8.17 Glossary :

1. **Collection Interface** – Provides the interfaces and classes to manage the group of data.
2. **List Interface** – The List interface is derive form collection interface. It allowed duplicated values to store. The List interface has implemented by ArrayList class, LinkedList class, Vector class and Stack class.
3. **Queue Interface** – The Queue interface work on the principal of Fist–in–First–out (FIFO). The Queue interface has implemented by Priority Queue class, LinkedList class and PriorityBlockingQueue class
4. **Set Interface** – The Set interface has implemented by HasSet class, TreeSet and LindedHasSet class.
5. **Map Interface** – The Map interface is parallel to collection interface. It works on the principal of key and value. The Map interface has implemented by HasMap class, LinkdedHasMap class and TreeMap

8.18 Assignment :

1. Write a note on Collection Interface.
2. Write a note on Queue Interface.
3. Write a note on Map Interface.

8.19 Activities :

1. Write a program to show the use of collection interface.
2. Write a program to show the use of Hasset class.
3. Write a program to show the use of Queue interface.
4. Write a program to show the use of Iterator.

8.20 Case Study :

1. Prepare the chart of Collection Interface with its child classes and interfaces.
2. Prepare the chart of Map Interface with its child classes and interfaces.

8.21 Further Reading :

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000.
2. Java 2, the Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999.
3. Programming with Java, Ed. 2, E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000.
4. The Java Tutorial, Ed. 2, 2 volumes, Mary Campione and Kathy Walrath, Addison Wesley Longmans, 1998.
5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele & Gilad Bracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000.
6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition, 2000

BLOCK SUMMARY :

In this collectively we have learned Object Oriented Concepts we can say that an object is an identifiable entity with some characteristic features and behavior. Anything which has some properties and performs some behavior is called an object. Keeping mind the programming for user's application is said as object oriented programming. In mean time we understood Classes which is nothing but a Collection of similar types of objects is called Class. A Class is also called as an Object Factory as once the class is created we can create as many objects as we wish using that class.

There is something called Java Method .A Java method is a collection of statements that are grouped together to perform an operation. A method definition consists of a method header and a method body like Modifiers, Return Type, and Method Name, Parameters, Method Body. Creating Objects,

Further we have learned that in general there are two different methods of passing an argument to a function. These methods are i) Call by value, ii. Call by reference. Call by value : In call by value method, the value of an argument is copied to the formal parameter. That is, the changes made in the actual argument are not reflected into the formal parameter, and Call by Reference, in Call by Reference method, the changes made to the actual argument are also reflected in the formal argument.

It is also understood that the constructors are not called in the

main( ) function. They are automatically executed at the time of object creation. For Example : the above program using default constructors can be written as DEFAULT CONSTRUCTOR. The default constructors are those constructors which do not accept any parameters/arguments. These constructors are automatically executed at the time of object creation. Another thing is related to Parameterised Constructors. The parameterised constructors are those constructors which accept parameters/arguments. At the time of object creation the values of these arguments/parameters are passed from main ( ) function.

There is learning about the 'this' keyword is used when a function will need to refer to the object which invoked it. The 'this' keyword can be used inside any method to refer to the current object. At the end of this Unit we came to know that the use of this keyword is redundant but perfectly correct. Inside rectangle ( ), this will always refer to the invoking object. When a local variable has the same name as an instance variable, the local variable hides the instance variable. That is why length and width were not used as the name of the parameters to the rectangle ( ) constructor inside the rectangle class. If we would have used in that way, then breadth would have referred to the formal parameter hiding the instance variable breadth

BLOCK ASSIGNMENT :

❖ **Short Questions :**

1. Explain call by value method.
2. What do you mean by call by reference method ?
3. What do you mean by instance variable static ?
4. What are static instance variables also called as ?
5. What ways is an interface similar to a class ?
6. What do mean by implementing interfaces.
7. What do you mean by an Object
8. Define Object Oriented Programming ?
9. Name the existing packages in Java.
10. What do mean by a class is placed in a package ?

❖ **Long Questions :**

1. Explain Java programming environment with the help of diagram.
2. Write a note on Java technology.
3. List the four categories of visibility for class members.
4. Write a note on instance variables ?
5. What are the results when a class is placed in a package ?
6. Write a note on call by reference method.
7. Explain in what ways is an interface similar to a class.
8. Write the rules for implementing interfaces.
9. Explain how you can convert numbers into string.
10. Write a program to demonstrate the use of parse Int.
11. What are the results when a class is placed in a package ?

❖ **Enrolment No. :** ☐

1. How many hours did you need for studying the units ?

Unit No.	5	6	7	8
No. of Hrs.				

2. Please give your reactions to the following items based on your reading of the block :

Items	Excellent	Very Good	Good	Poor	Give specific example if any
Presentation Quality	☐	☐	☐	☐	________
Language and Style	☐	☐	☐	☐	________
Illustration used (Diagram, tables etc)	☐	☐	☐	☐	________
Conceptual Clarity	☐	☐	☐	☐	________
Check your progress Quest	☐	☐	☐	☐	________
Feed back to CYP Question	☐	☐	☐	☐	________

3. Any other Comments

....

....

....

....

....

....

....

....

Dr. Babasaheb Ambedkar Open University Ahmedabad

BCAR-204/ DCAR-204

Object Oriented Concepts & Programming–1 (Core Java)

BLOCK 3 : INHERITANCE, EXCEPTION HANDLING AND MULTITHREADING

INHERITANCE, EXCEPTION HANDLING AND MULTITHREADING

Block Introduction :

This block is last block of this book and has further detail study about Inheritance. Java Inheritance defines a relationship between a superclass and its subclasses. In this you study first Concept of Inheritance, Polymorphism and Final Keyword. After this you understand about Exception handling which deals with Try and Catch Block, Multiple Catch Statements, Methods defined by Exception, Throw able, Exceptions, Throws/throw Keyword, Using Finally Keyword and Nested Try Statements. After this there is last Unit which makes you understand about Utilities & Multithreading. This will be having rest of the details such as Comparing Arrays, Creating a Hash Table Multithreading, Thread Life Cycle, the Thread Class and the Runnable Interface, Thread Priorities, Synchronization, Deadlock, Suspending, Resuming and Stopping Threads

Block Objectives :

After learning this block, you will be able to :

- Explain the concept of inheritance
- Define Polymorphism
- Compile time and Runtime
- Illustrate super keyword and final keyword
- Explain the types of exception and uncaught exception
- Use try and catch blocks, multiple catch statements
- Illustrate methods defined by exception and throw able
- Discuss defined exceptions
- Describe throws/ throw keyword
- State finally keyword and nested try statements
- Identify utility package
- Describe arrays and hash table
- Explain the thread lifecycle
- Define thread class and runnable interface
- State thread priorities
- Point out synchronisation
- Illustrate deadlock
- Explain suspending, resuming and stopping threads

Block Structure :

Unit 09

INHERITANCE

UNIT STRUCTURE

9.0 Learning Objectives :

After learning this unit, you will be able to :

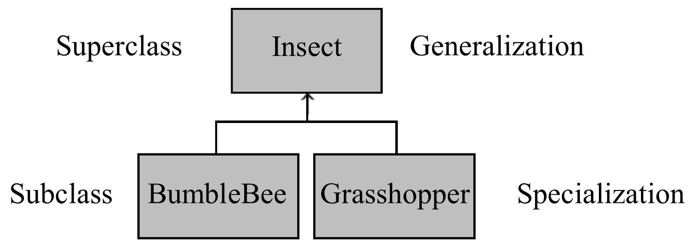
- Explain the concept of inheritance
- Define Polymorphism
- Compile time and Runtime
- Illustrate super keyword and final keyword

9.1 Introduction :

Inheritance can be defined as the process where one object acquires the properties of another. With the use of inheritance, the information is made manageable in a hierarchical order.

Java Inheritance defines a relationship between a superclass and its subclasses. This means that an object of a subclass can be used wherever an object of the superclass can be used. Class Inheritance in java mechanism is used to build new classes from existing classes. The inheritance relationship is transitive: if class X extends class Y, then A class Z, which extends class X, will also inherit from class Y.

9.2 Concept of Inheritance :



Concept of Inheritance

Figure 9.1 Concept of Inheritance

"Inheritance is one of the cornerstones of OOP because it allows for the creation of hierarchical classifications. Using inheritance, you can create a general class that defines traits common to a set of related objects, that is, objects with common attributes and behaviours. This class may then be inherited by other, more specific classes", each adding only those attributes and behaviours that are unique to the inheriting class.

❖ **Need of Inheritance :**

The various needs of inheritance are given below :

1. Closer to Real–World
2. Code Reusability
3. Transitive Nature

As stated earlier, inheritance leads to the definition of generalized classes that are at the top of an inheritance hierarchy, thus it is an implementation of generalization. This feature available in C++ makes the data and methods of a Superclass or base class available to its subclass or derived class. It has many advantages, the most important of that is the reusability of code. Once a class has been created, it can be used to create new subclasses.

❖ **Generalisation / Specialisation :**

A class that is inherited is referred to as a base class. The class that does the inheriting is referred to as the derived class. Each instance of a derived class includes all the members of the base class. The derived class inherits all the properties of the base class. Therefore, the derived class has a large set of properties than its base class. However, a derived class may override some of the properties of the base class.

To inherit a class, the definition of one class can be incorporated into another by using the extends keyword. This class can then be inherited by other, more specific classes, each adding those things that are unique to it. The syntax of inheriting class is given below:

❖ **Syntax :**

<access specifier> class <class name(Subclass)> extends <class name(Superclass)>

For example,

Public class B extends A

Public class C extends B

Public Class D extends C

The public data members and methods (Except constructors) in the superclass are inherited by the subclass, i.e., their definitions are copied into the subclass's class definition. No members of the subclass are visible to the superclass.

Now let us consider an example to illustrate the same:

```
//Create a superclass
Class SuperClass
{
   int x,y;
```

```
    void showXY( )
    {
    System.out.println ("x and y" + x + " " + y);
    }
}
```

//Create a subclass by extending class A

```
Class SubClass extends SuperClass
{
    int z;
    void showZ( )
    {
    System.out.println ("z is:" + z);
    }

    Void sum( )
    {
    System.out.println ("x+y+z" + (x + y+ z));
    }
    }
Class DemoInheritance
{
    public static void main (String args[])
{
    SuperClasssuperob = new SuperClass ( );
    SubClasssubob= new SubClass ( );
    //The superclass can refer itself
    superob.x=21;
    superob.y=4;
    System.out.println ("Contents of Superclass")
    superob.showXY( );
```

//The subclass has access to all public members of its superclass

```
subob.x=30;
subob.y=11;
subob.z=13;
System.out.println ("Contents of subclass")
subob.showXY( )
subob.showZ( );
System.out.println ( );
System.out.println ("Sum of x, y and z in subclass");
```

```
subob.sum( );
}
}
```

The output of above program is given below:

```
Contents of superclass
X and y 21 4
Contents of subclass
x and y 30 11
z is: 13
Sum of x, y and z in subclass
```

As it is given in the above program, the subclass SubClass includes all of the members of its superclass SuperClass. That is why subob can access x and y and call showXY( ). Also, inside add ( ), x and y can be referred to directly, as if they were part of SubClass. Although a subclass includes all of the members of its superclass, it cannot access members of the superclass that have been declared as private.

❑ **Check Your Progress – 1 :**

1. Write the syntax of inheriting a class.
2. Explain Implicit Subclass to Super class Conversion with the help of an example.

....

....

....

....

....

9.3 Polymorphism :

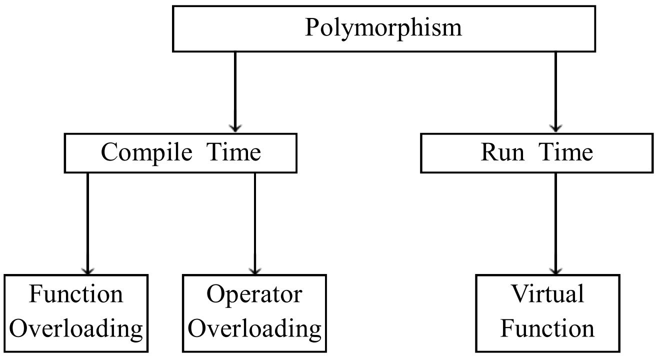


Figure 9.2 Polymorphism

"Polymorphism allows one interface to be used for a set of actions i.e., one name may refer to different functionality. Polymorphism allows an object to accept different requests of a client (it then properly interprets the request like choosing appropriate method) and responds according to the current state of the runtime system, all without bothering the user."

There are two types of polymorphism:

1. Compile–time polymorphism
2. Runtime Polymorphism

1. Compile time Polymorphism :

In compile time Polymorphism, method to be invoked is determined at the compile time. Compile time polymorphism is supported through the method overloading concept in java.

Method overloading means having multiple methods with same name but with different signature (number, type and order of parameters).

Here is the code of the example:

```
class One
{
public void funOne (int a)
{
System.out.println ("The value of class A is:" + a);
}
public void funOne (int a, int b)
{
System.out.println ("The value of class B is:" + a + "and" + b);
}
}
public class PolyOne
{
public static void main (String [ ] args)
{
One obj=new One ( );
//Here compiler decides that funOne (int) is to be called and "int" will
be printed.
obj.funOne (20);
//Here compiler decides that funOne (int, int) is to be called and "int
and int" will be printed.
obj.funOne (20, 30);
}
}
```

The output of above program is given below:

```
The value of class A is: 20
The value of class B is: 20 and 30
```

2. Runtime Polymorphism :

In runtime polymorphism, the method to be invoked is determined at the run time. The example of run time polymorphism is method overriding which is also called dynamic method dispatch is explained below :

❖ **Method Overriding :**

If a class inherits a method from its super class, then there is a chance to override the method provided that it is not marked final.

Overriding means redefining a method in an inheritance hierarchy. In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass then the method in the subclass is said to override the method in the superclass.

The benefit of overriding is: ability to define a behavior that's specific to the sub class type which means a subclass can implement a parent class method based on its requirement.

In object oriented terms, overriding means to override the functionality of any existing method.

Example :

Let us look at an example.

```
class Animal
{
public void eat ( )
{
System.out.println ("Animals can eat");
}
}
Class Dog extends Animal
{
public void eat ( )
{
System.out.println ("Dog can eat and drink");
}
}
public class TestCat
{
public static void main (String args [ ])
{
Animal a = new Animal ( ); //Animal reference and object
Animal b= new Dog ( ); //Animal reference but Rat object
a.eat ( ); //runs the method in Animal class
b.eat ( ); //runs the method in Dog class
}
}
```

This will produce the given output:

```
Animals can eat
Dog can eat and drink
```

In the example given above, you can see that even though Dog is a type of Animal it runs the eat method in the Dog class. The reason for this is: In compile time the check is made on the reference type. However, in the runtime, JVM figures out the object type and would run the method that belongs to that particular object.

Therefore, in the above example, the program will compile properly since Animal class has the method eat. Then at the runtime it runs the method specific for that object.

❖ **Rules for method overriding :**

- "The return type should be the same or a subtype of the return type declared in the original overridden method in the super class.
- A method declared final keyword example required cannot be overridden.
- The access level cannot be more restrictive than the overridden method's access level. For example : if the super class method is declared public then the overriding method in the sub class cannot be either private or public. However the access level can be less restrictive than the overridden method's access level.
- The argument list should be exactly the same as that of the overridden method.
- Instance methods can be overridden only if they are inherited by the subclass.
- A method declared static cannot be overridden but can be re–declared.
- If a method cannot be inherited then it cannot be overridden.
- A subclass in a different package can only override the non–final methods declared public or protected.
- An overriding method can throw any uncheck exceptions, regardless of whether the overridden method throws exceptions or not. However, the overridden method should not throw checked exceptions that are new or broader than the ones declared by the overridden method. The overriding method can throw narrower or fewer exceptions than the overridden method.
- A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
- Constructors cannot be overridden."

❖ **Using the super keyword :**

When invoking a superclass version of an overridden method the super keyword is used.

```
class Animal
{
public void move ( )
{System.out.println ("Animals can move");}
}//Animal
Class cow extends Animal
{
public void move ( )
{
```

```
super.move ( );   //invokes the super class method
System.out.println ("Cow can walk and run"):
}
public class TestCow
{
public static void main (String args [ ])
{
Animal b = new Cow ( ); //Animal reference but Cow object
b.move ( ); //runs the method in Cow class
}
}
```

This would produce following result:

```
Animals can move
Cow can walk and run
```

❑ **Check Your Progress – 2 :**

1. Write the rules for method overriding.
2. What do you mean by the term polymorphism

....

....

....

....

....

9.4 Final Keyword :

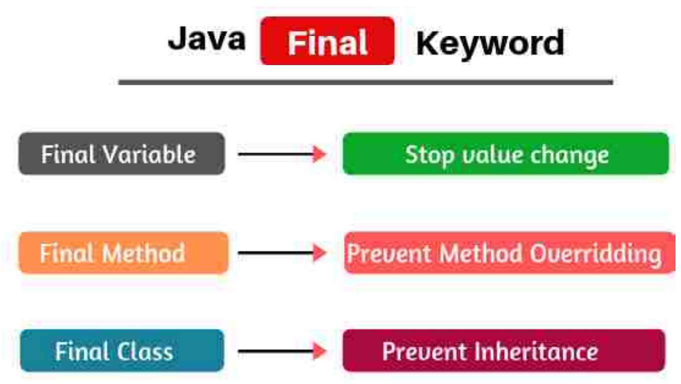


Figure 9.3 Final Keyword

The final keyword has mainly three uses :

1. Creating constants
2. Preventing method overriding
3. Preventing inheritance

Let us discuss these uses in order to understand the concept of final keyword in detail :

1. Creating Constants :

Declaring a variable as final makes it constant, doing so prevents the contents from being modified. This means that you must initialize a final variable when it is declared.

For example :

Final int FILE_NEW=1

Final int FILE_OPEN=2

Subsequent parts of your program can now use FILE_OPEN, FILE_NEW etc. as if they were constants. It is a common naming convention to choose all uppercase identifiers for final variables. Thus, a final variable is essentially a constant.

2. Preventing method overriding :

The keyword final can also be applied to methods but its meaning is substantially different than when it is applied to variables. Methods declared as final cannot be overridden, that is, a method in the superclass cannot be overridden in the subclass.

3. Preventing inheritance :

In some cases, you may want to prevent a class form being inherited. To do this, precede the class declaration with final. Declaring a class as final implicitly declares all of its methods a final, too. It is illegal to declare a class both abstract and final since an abstract class is incomplete by itself and relies upon its subclass to provide complete implementations.

❑ **Check Your Progress – 3 :**

1. List the uses of final keyword.
2. Write the syntax of final keyword

....

....

....

....

....

3. ________ can be defined as the process where one object acquires the properties of another.

   (A) Object (B) Inheritance (C) Method (D) Function

4. A class that is inherited is referred to as a ________ class.

   (A) base (B) super (C) sub (D) none of these

5. Polymorphism allows one ________ to be used for a set of actions

   (A) Method (B) Class (C) Interface (D) Function

6. In compile time Polymorphism, method to be ________ is determined at the compile time.

   (A) Inherited (B) Java class (C) Access (D) Invoked

7. ________ methods can be overridden only if they are inherited by the subclass.

   (A) Instance (B) Interface (C) Private (D) None of these

9.5 Let Us Sum Up :

This Unit No.1 of this Block we have understood "Inheritance is one of the cornerstones of OOP because it allows for the creation of hierarchical classifications. Using inheritance, you can create a general class that defines traits common to a set of related objects", that is, objects with common attributes and behaviours. The various needs of inheritance are

1. Closer to Real–World,
2. Code Reusability
3. Transitive Nature.

The study of Generalisation/Specialisation has made us understand Syntax, Create a superclass then Create a subclass by extending class A and also the subclass has access to all public members of its superclass. Then further studied that "Polymorphism allows one interface to be used for a set of actions i.e. one name may refer to different functionality. Polymorphism allows an object to accept different requests of a client (it then properly interprets the request like choosing appropriate method) and responds according to the current state of the runtime system", all without bothering the user. There are two types of polymorphism, 1. Compile–time polymorphism, 2. Runtime Polymorphism.

We understood Method Overriding, if a class inherits a method from its super class, then there is a chance to override the method provided that it is not marked final. Overriding means redefining a method in an inheritance hierarchy. In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass then the method in the subclass is said to override the method in the superclass. We have also understood in detail all Rules for method overriding. There is also good understanding about the final keyword has mainly three uses 1. Creating constants, 2. Preventing method overriding, and 3. Preventing inheritance

9.6 Suggested Answer for Check Your Progress :

- **Check Your Progress 1 :**

  See Section 9.2

- **Check Your Progress 2 :**

  See Section 9.3

- **Check Your Progress 3 :**

  1 : See Section 9.4 **2 :** See Section 9.4

  3 : B **4 :** A **5 :** C **6 :** D **7 :** A

9.7 Glossary :

1. **Inheritance –** Inheritance is one of the cornerstones of OOP because it allows for the creation of hierarchical classifications.
2. **Overriding –** Overriding means redefining a method in an inheritance hierarchy. In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass then the method in the subclass is said to override the method in the superclass.

9.8 Assignment :

Explain how a subclass has access to all public members of its superclass.

Write a program to create a superclass.

9.9 Activities :

Write a program to explain inheritance

9.10 Case Study :

Create a class Medicine to represent a drug manufactured by a pharmaceutical company. Provide a function display Label() in this class to print Name and address of the company.

Derive Tablet, Syrup and Ointment classes from the Medicine class. Override the display Label() function in each of these classes to print additional information suitable to the type of medicine. For example, in case of tablets, it could be "store in a cool dry place", in case of ointments it could be "for external use only" etc.

Create a class Test Medicine. Write main function to do the following:

1. Declare an array of Medicine references of size 10
2. Create a medicine object of the type as decided by a randomly generated integer in the range 1 to 3.
3. Refer Java API Documentation to find out random generation feature.
4. Check the polymorphic behaviour of the display Label() method.

9.11 Further Reading :

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000
2. Java 2, The Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999
3. Programming with Java, Ed. 2, E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000
4. The Java Tutorial, Ed. 2, 2 volumes,MaryCampione and Kathy Walrath, Addison Wesley Longmans, 1998
5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele &GiladBracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000
6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition, 2000

Unit 10

EXCEPTION HANDLING

UNIT STRUCTURE

10.0 Learning Objectives :

After learning this unit, you will be able to :

- Explain the types of exception and uncaught exception
- Use try and catch blocks, multiple catch statements
- Illustrate methods defined by exception and throwable
- Discuss defined exceptions
- Describe throws/ throw keyword
- State finally keyword and nested try statements

10.1 Introduction :

A runtime error that arises during the execution of a program is called an exception. Various languages don't support handling exceptions and thus, the errors are to be checked for and taken care of manually through error codes etc. This approach is quite cumbersome and troublesome.

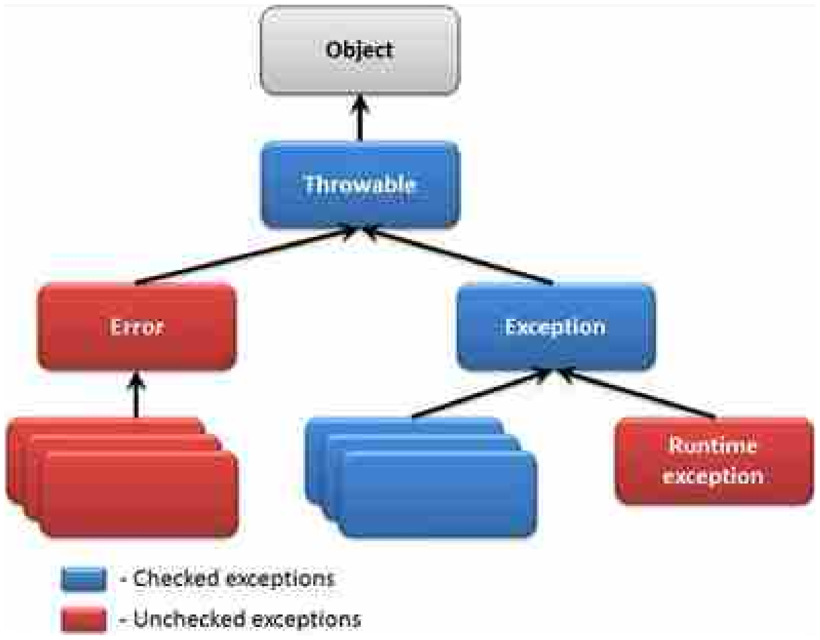


Figure 10.1 Exception Handling

An exception can occur for many different reasons, including the following:

- A user has entered invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications, or the JVM has run out of memory.

Such exceptions can often be caused due to programmer or user errors and sometimes due to other physical resources failing in some manner.

10.2 Types of Exceptions :

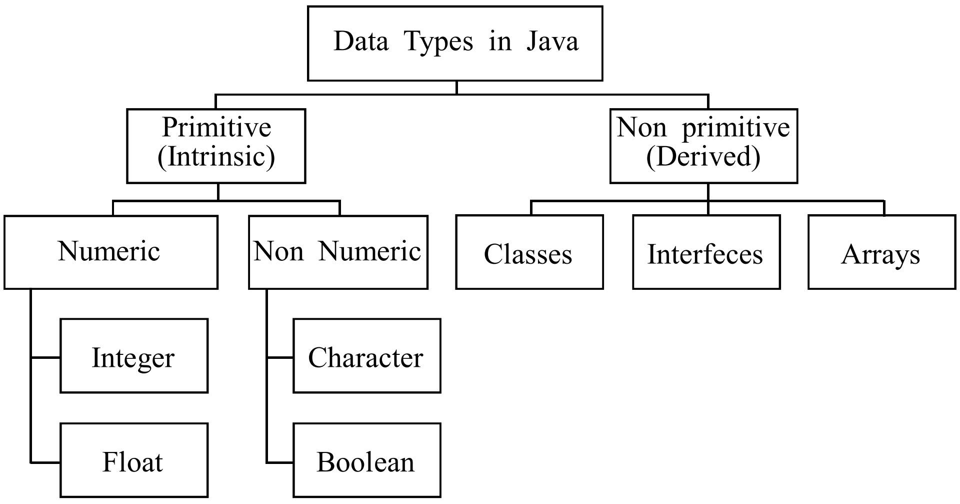


Figure 10.2 Data type in Java

Java provides several predefined Exception classes in the package java.lang. In order to understand the concept of exception handling, you have to understand the following categories of exceptions:

1. Checked exceptions
2. Unchecked exceptions
3. Errors

The checked exceptions must be caught or re–thrown. The unchecked exceptions do not have to be caught.

1. **Checked exceptions :** This is basically a user error that the programmer cannot foresee. For example, if a file is to be opened but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.
2. **Unchecked or Runtime exceptions :** This exception could have probably been avoided by the programmer. Runtime errors are ignored during compilation as opposed to checked exceptions.
3. **Errors :** These are not exceptions at all but problems that arise beyond the control of the user or the programmer. Usually since the programmer can rarely do anything about an error it is ignored. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

❖ **Exception Hierarchy :**

All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class.

Normally errors cannot be trapped from the Java program. Such conditions arise in the case of severe failures which java programs do not handle. Errors are generated to indicate errors generated by the runtime environment. Example: JVM is out of Memory. Normally programs cannot recover from errors.

The Exception class has two main subclasses : IOException class and Runtime Exception Class.

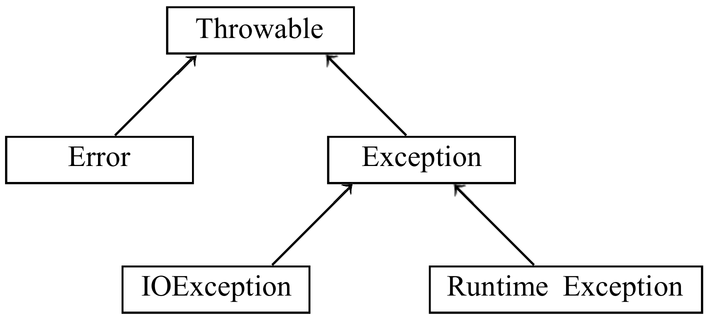


Figure 10.3 Exception Hierarchy

❖ **Common Exceptions :**

In java it is possible to define two categories of Exceptions and Errors.

- JVM Exceptions: These errors are either exclusively or logically thrown by the JVM. Examples :
  - NullPointerException,
  - ArrayIndexOutOfBounds Exception,
  - ClassCastException and many more
- Programmatic exceptions. These exceptions are thrown explicitly by the application or the API programmers Examples:
  - IllegalArgumentException,
  - IllegalStateException

❑ **Check Your Progress – 1 :**

1. Name the subclasses of exception class.

....

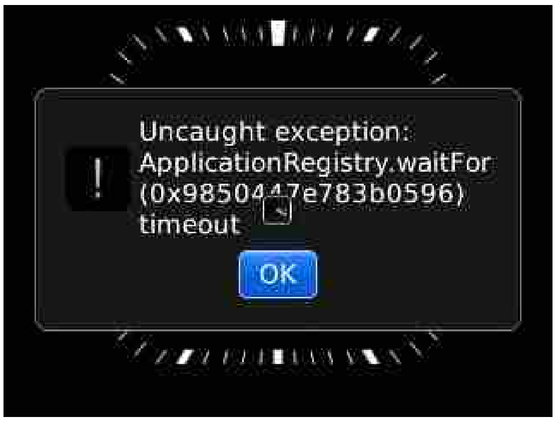
....

....

....

....

10.3 Uncaught Exception :



Uncaught exception:
ApplicationRegistry.waitFor
(0x9850447e783b0596)
timeout

Figure 10.4 Uncaught Exception

When the exceptions are not caught in a try/catch block, then what you often see in practice is Java prints the exception stack trace and then terminate your program. "Java actually handles uncaught exceptions according to the thread in which they occur. When an uncaught exception occurs in a particular thread, Java looks for what is called an uncaught exception handler, actually an implementation of the interface UncaughtExceptionHandler". The latter interface has a method handleException (), which the implementer overrides to take appropriate action, such as printing the stack trace to the console.

The specific procedure is as follows. When an uncaught exception occurs, the JVM does the following:

- It calls a special private method, dispatchUncaughtException(), on the Thread class in which the exception occurs
- It then terminates the thread in which the exception occurred

The dispatch Uncaught Exception method, in turn, calls the thread's get Uncaught Exception Handler () method to find out the appropriate uncaught exception handler to use. Normally, this will actually be the thread's parent Thread Group, whose handle Exception () method by default will print the stack trace.

Thus, the full process used to determine which uncaught exception handler is called is shown in Figure below :

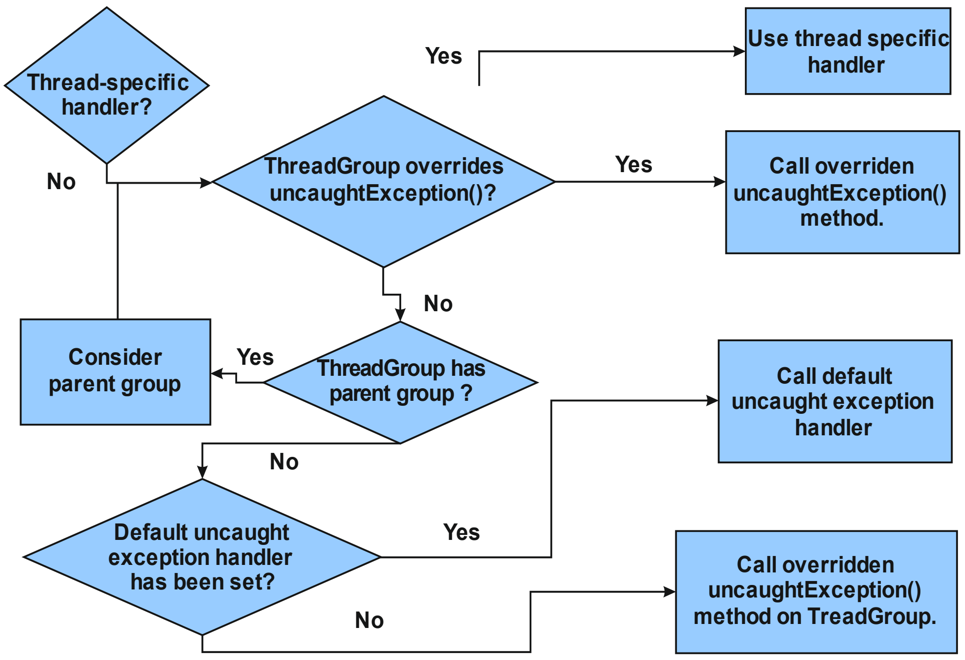

Thread-specific handler?
Yes
Use thread specific handler
No
ThreadGroup overrides uncaughtException()?
Yes
Call overriden uncaughtException() method.
No
Consider parent group
Yes
ThreadGroup has parent group ?
Call default uncaught exception handler
No
Default uncaught exception handler has been set?
Yes
Call overridden uncaughtException() method on TreadGroup.
No

Figure 10.5 Process by which Java decides on which uncaught exception handler to call for a given thread

❑ **Check Your Progress – 2 :**

1. Explain an uncaught exception.
2. Write a program for an uncaught exception handler.

....

....

....

....

....

10.4 Using Try and Catch Block :

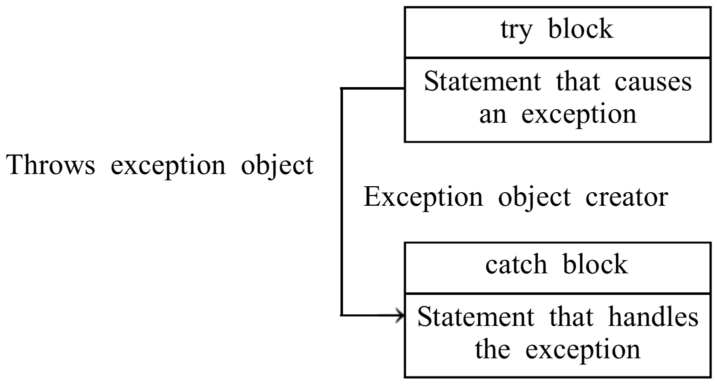

try block
Statement that causes an exception
Throws exception object
Exception object creator
catch block
Statement that handles the exception

Figure 10.6 Try and Catch Block

When an exception arises, an object representing that exception is created and thrown in the method that caused the error. This mechanism in Java creates an object which describes an exception condition.

When an exception condition arises, an object representing that exception is created and thrown in the method which caused the error. When an error occurs within a method, the method creates an object and hands it off to the runtime system.

This exception object is called an exception object which contains information about the error along with its type and the state of the program when the error occurred. This method of creating an exception object and handing it to the runtime system is called throwing an exception.

Java Exception handling is managed using five keywords –

(a) try

(b) catch

(c) throw

(d) throws

(e) finally

The general form of the Exception Handling is

```
try
{
//do something that might cause an exception
}
catch (ExceptionType variable)
{
//handle the exception
}
finally
{
//always execute these statements
}
```

The program statements which you want to monitor for exceptions are written within try block. If an exception occurs within the try block (when exception is thrown), it has to be handled in some manner in catch block.

Any code which exactly has to be executed, after exiting from try clock is put in finally block. The catch block is only executed if a particular exception occurs. Whereas, the finally block is always executed, irrespective of exception occurs or not. A brief description of these keywords is given below:

❖ **Try :**

The code that could generate errors is put in try block. The statements are executed unless an exception occurs. If an exception is thrown, java breaks out of the try block by skipping the rest of the statements and searches for a respective matching catch statement.

If an exception does not occur, java executes all the statements within that block.

❖ **Catch :**

A catch statement is included immediately following the try block; it specifies the exception type that has to be caught.

❖ **Finally :**

It creates a block of code following the try catch block. It will execute whether or not an exception is thrown. Each try statement should have at least one catch or finally clause.

❑ **Check Your Progress – 3 :**

1. List the keywords that manage Java exception handling.

....

....

....

....

....

10.5 Using Multiple Catch Statements :

A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following:

```
try
{
   //Protected code
}catch(ExceptionType1 e1)
{
   //Catch block
}catch(ExceptionType2 e2)
{
   //Catch block
}catch(ExceptionType3 e3)
{
   //Catch block
}
```

The previous statements demonstrate three catch blocks but you can have any number of them after a single try. The exception is thrown to the first catch block in the list if it occurs in the protected code. If the data type of the exception thrown matches ExceptionType1, it gets caught there. If not, the exception passes down to the second catch statement.

This continues until the exception either is caught or falls through all catches, in which case the current method stops execution and the exception is thrown down to the previous method on the call stack.

❖ **Catching Exceptions :**

By using a combination of the try and catch keywords a method catches an exception. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code and the syntax for using try/catch looks like the following :

```
try
{
   //Protected code
}catch(ExceptionName e1)
{
   //Catch block
}
```

A catch statement comprises of declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block (or a block) that follows the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter.

❑ **Check Your Progress – 4 :**

1. Write the syntax for multiple catch blocks.
2. Explain the execution of multiple catch statements.

....

....

....

....

....

10.6 Using Methods Defined by Exception and Throwable :

Following is the list of important methods available in the Throwable class :

SN	Methods with Description
1	"public String getMessage() Returns a detailed message about the exception that has occurred. This message is initialised in the Throwable constructor.
2	Public Throwable getCause() Returns the cause of the exception as represented by a Throwable object.
3	public String toString() Returns the name of the class concatenated with the result of get Message()
4	public void printStackTrace() Prints the result of to String() along with the stack trace to System. err, the error output stream.
5	Public Stack Trace Element [] getStackTrace() Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack and the last element in the array represents the method at the bottom of the call stack.

6	Public Throwable fillInStackTrace() Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace".

❑ **Check Your Progress – 5 :**

1. Write a note on public Throwable get Cause().
2. Write a program that throws an exception.

....

....

....

....

....

10.7 User Defined Exceptions :

"You can create your own exceptions in Java. Keep the following points in mind while writing your own exception classes:

- All exceptions must be a child of Throwable.
- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
- If you want to write a runtime exception, you need to extend the Run time Exception class."

We can define our own Exception class as below :

```
class MyException() extends Exception{
}
```

You just need to extend the Exception class to create your own Exception class. These are considered to be checked exceptions. The following Insufficient Funds Exception class is a user–defined exception that extends the Exception class, making it a checked exception. An exception class is like any other class, containing useful fields and methods.

Example :

```
// File Name InsufficientFundsException.java
import java.io.*
public class InsufficientFundsException extends Exception
{
private double amount
public InsufficientFundsException(double amount)
{
this.amount = amount
}
public double getAmount()
{
return amount
}
}
```

To demonstrate using our user–defined exception, the following CheckingAccount class contains a withdraw () method that throws an InsufficientFundsException.

```
// File Name CheckingAccount.java
import java.io.*;
public class ChkAccount
{
private double balance;
private intAcNumber;
public ChkAccount(intAcNumber)
{
this.AcNumber = AcNumber;
}
public void deposit(double amount)
{
balance += amount;
}
public void withdraw(double amount) throwsInSufficientFundsException
{
if(amount <= balance)
{
balance –= amount;
}
else
{
double needs = amount – balance
throw new InSufficientFundsException(needs)
}
}
public double getBalance()
{
return balance;
}
public intgetAcNumber()
{
return AcNumber;
}
}
```

The following BankDemo program demonstrates invoking the deposit () and withdraw () methods of CheckingAccount.

```
// File Name BankDemo.java public class BankDemo
{
public static void main(String [] args)
{
ChkAccount c = new ChkAccount(101);
System.out.println ("Depositing Rs500..."); c.deposit(500.00);
try
{
System.out.println("\nWithdrawing Rs100...")
c.withdraw(100.00);
System.out.println("\nWithdrawing Rs600...")
c.withdraw(600.00)
}catch(InsufficientFundsException e)
{
System.out.println("Sorry but you are short Rs".
          + e.getBalance());
e.printStackTrace();
}
}
}
classInSufficientFundsException extends Exception
{
InSufficientFundsException( double needs)
{
System.out.println("Insufficient fund"+ needs);
}
}
```

❑ **Check Your Progress – 6 :**

1. What points should be kept in mind when writing your own exception classes ?

....

....

....

....

....

10.8 Using Throws/Throw Keywords :

Before catching an exception it is must to be thrown first. This means that there should be a code somewhere in the program that could catch the exception. We use throw statement to throw an exception or simply use the throw keyword with an object reference to throw an exception. A single argument is required by the throw statement i.e. a throwable object. Throwable objects are instances of any subclass of the Throwable class.

throw new VeryFastException() Exception

Handling

The reference should be of type Throwable or one of its subclasses.

For instance the example below shows how to throw an exception. Here we are trying to divide a number by zero so we have thrown an exception here as

"throw new MyException ("can't be divided by zero")"

```
classMyException extends Exception {
public MyException(String msg){
super(msg);
}
}
public class Test {
static void divide(intfirst,int second) throws MyException{
    if(second==0)
    throw new MyException("can't be divided by zero") }
  public static void main(String[] args) {
    try {
  divide(4,0);
    }
  catch (MyExceptionexc) {
  exc.printStackTrace();
    }
    }//main
  }//Test
```

❖ **Output :**

```
C:\Computer\vinod\Exception>javac Test.java
C:\Computer\vinod\Exception>java Test
MyException: can't be divided by zero
at Test.divide (Test.java:10)
at Test.main(Test.java:15)
```

The method must declare by using the throws keyword if it does not handle a checked exception. The throws keyword appears at the end of a method's signature.

By using the throw keyword you can throw an either newly instantiated exception or an exception you just caught. The difference in throws and throw keywords should be understood.

A method can declare that it throws more than one exception, in which case the exceptions are declared in a list separated by commas. For example, the following method declares that it throws a Remote Exception and an Insufficient Funds Exception.

```
import java.io.*;
public class className
{
public void withdraw(double amount) throws
RemoteException,InsufficientFundsException
   {
      // Method implementation
   }
   //Remainder of class definition
}
```

Difference between throw and throws keywords

We use the throw keyword in order to force an exception. The throw keyword (note the singular form) is used to force an exception. The throw keyword also passes a custom message to your exception handling module. For instance, in the above example, we have used

```
Throw new MyException ("can't be divided by zero")
```

Whereas, we use the throws keyword when we already know that a particular exception will be thrown or when we pass a possible exception. Point to note here is that the Java compiler very well knows about the exceptions thrown by some methods so it insists us to handle them.

We can also use throws clause on the surrounding method instead of try and catch exception handler. For instance in the above given program, we have used the following clause which will pass the error up to the next level

```
static int divide(int first, int second) throws MyException{
```

❑ **Check Your Progress – 7 :**

1. Where is throw keyword declared in a method ?
2. Write a program using throw keyword.

....

....

....

....

....

10.9 Using Finally Keyword :

"The finally keyword is used to create a block of code which follows a try block. A finally block of code always executes, whether or not an exception has occurred."

Using a finally block allows you to run any cleanup–type statements that you want to execute, no matter what happens in the protected code.

A finally block appears at the end of the catch blocks and has the following syntax:

```
try
{
   //Protected code
}catch(ExceptionType1 e1)
{
   //Catch block
}catch(ExceptionType2 e2)
{
   //Catch block
}catch(ExceptionType3 e3)
{
   //Catch block
}finally
{
   //The finally block always executes.
}
```

Note the following :

- A catch clause cannot exist without a try statement.
- It is not compulsory to have finally clauses whenever a try/catch block is present.
- The try block cannot be present without either catch clause or finally clause.

Any code cannot be present in between the try, catch, finally blocks.

❑ **Check Your Progress – 8 :**

1. Write the syntax of finally keyword.
2. Write some points for finally keyword.

....

....

....

....

....

10.10 Nested Try Statements :

In Java we can have nested try and catch blocks. It means that, a try statement can be inside the block of another try. If an inner try statement does not have a matching catch statement for a particular exception, the control is transferred to the next try statement's catch handlers that are expected for a matching catch statement. This continues until one of the catch statements succeeds, or until the entire nested try statements are done in. If no one catch statements match, then the Java run–time system will handle the exception.

The syntax of nested try–catch blocks is given below:

```
try {
    try {
    // ...
    }
    catch (Exception1 e)
    {
    //statements to handle the exception
    }
}
catch (Exception e2)
{
//statements to handle the exception
}
```

Example

```
import java.io.*;
public class NestedTryDemo{
    public static void main (String args[])throws IOException {
        int number=5,result=0;
try{
FileInputStream fis=null;
fis = new FileInputStream (new File (args[0]))
try{
result=number/0;
System.out.println("The result is"+res);
}
catch(ArithmeticException e){
System.out.println("divided by Zero");
}
}
catch (FileNotFoundException e){
System.out.println("File not found!")
}
catch(ArrayIndexOutOfBoundsException e){
System.out.println("Array index is Out of bound!Argument required"); }
catch(Exception e){
System.out.println("Error.."+e)
}
}
}
```

❖ **Output :**

```
C:\Computer\>javac NestedTry.java
C:\Computer\>java NestedTry
Array index is Out of bound! Argument required
```

In this given example, we have implemented nested try–catch blocks concept where an inner try block is kept with in an outer try block, that's catch handler will handle the arithmetic exception. But before that an Array Index Out Of Bounds Exception will be raised, if a file name is not passed as an argument while running the program.

❑ **Check Your Progress – 9 :**

1. Write the syntax of nested try catch block.
2. Explain the execution of nested try statements.

....

....

....

....

....

3. A runtime error that arises during the execution of a program is called an ________.

   (A) Exception (B) Error (C) occurrence (D) problem

4. A network connection has been lost in the middle of communications, or the ________ has run out of memory.

   (A) JDK (B) JRE (C) JAVA (D) PATH

5. ________ Exceptions is not occurred by JVM

   (A) NullPointerException,

   (B) ArrayIndexOutOfBounds Exception,

   (C) ClassCastException

   (D) File not found

6. Java Exception handling is managed using ________ keywords

   (A) One (B) Two (C) Four (D) Five

7. A ________ statement is included immediately following the try block; it specifies the exception type that has to be caught.

   (a) Try (b) Catch (c) Finally (d) Block

10.11 Let Us Sum Up :

Deals with An exception is a problem that arises during the execution of a program. It is a runtime error. In some of the languages, which do not support exception handling, errors must be checked and handled manually typically through the use of error codes and so on. This approach is quite cumbersome and troublesome.

Then there is understanding relating to the predefined Exception, Java provides several predefined Exception classes in the package java.lang. To understand how exception handling works in Java, you need to understand these categories of exceptions:

1. Checked exceptions
2. Unchecked exceptions
3. Errors

"When the exceptions are not caught in a try/catch block, then what you often see in practice is that Java prints the exception stack trace and then terminates your program. Java actually handles uncaught exceptions according to the thread in which they occur. When an uncaught exception occurs in a particular thread, Java looks for what is called an uncaught exception handler, actually an implementation of the interface Uncaught Exception Handler. The latter interface has a method handle Exception (), which the implementer overrides to take appropriate action, such as printing the stack trace to the console."

We have understood that Java Exception handling is managed using five keywords – a. try, b. catch, c. throw, d. throws and finally e. finally.

Catching Exceptions: "A method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code"

User defined Exceptions: While creating your own exceptions in Java, the following points are to be kept in mind.

- All exceptions must be a child of Throwable.
- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
- If you want to write a runtime exception, you need to extend the Runtime Exception class.

Using Throws/throw Keywords: "This means that there should be a code somewhere in the program that could catch the exception. We use throw statement to throw an exception or simply use the throw keyword with an object reference to throw an exception. Know after that we understood. The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred."

Using a finally block allows you to run any cleanup–type statements that you want to execute, no matter what happens in the protected code.

We have also understood about NESTED TRY STATEMENT In Java we can have nested try and catch blocks. It means that, a try statement can be inside the block of another try. If an inner try statement does not have a matching catch statement for a particular exception, the control is transferred to the next try statement's catch handlers that are expected for a matching catch statement. This continues until one of the catch statements succeeds, or until the entire nested try statements are done in. If no one catch statements match, then the Java run–time system will handle the exception

10.12 Suggested Answer for Check Your Progress :

- **Check Your Progress 1 :**
  See Section 10.2
- **Check Your Progress 2 :**
  See Section 10.3
- **Check Your Progress 3 :**
  See Section 10.4
- **Check Your Progress 4 :**
  See Section 10.5
- **Check Your Progress 5 :**
  See Section 10.6
- **Check Your Progress 6 :**
  See Section 10.7
- **Check Your Progress 7 :**
  See Section 10.8
- **Check Your Progress 8 :**
  See Section 10.9
- **Check Your Progress 9 :**
  1 : See Section 10.9 **2 :** See Section 10.9
  3 : A **4 :** B **5 :** D **6 :** D **7 :** B

10.13 Glossary :

1. **Uncaught Exceptions** – When the exceptions are not caught in a try/ catch block, then what you often see in practice is Java prints the exception stack trace and then terminates your program.
2. **Catching Exceptions –** A method catches an exception using a combination of the try and catch keywords.
3. **Throw an Exception –** This means that there should be a code somewhere in the program that could catch the exception. We use throw statement to throw an exception or simply use the throw keyword with an object reference to throw an exception.

10.14 Assignment :

Write a program which accepts two integers and an operator from the user. Perform the operation and fire the following user defined exception for following situations and handle the exceptions with proper error messages:

a. If the numbers are not of integer datatype, fire an exception

b. If the result is negative, fire an exception

c. Handle all the possible exceptions

10.15 Activities :

1. Write a note on catching exceptions.
2. Explain try and catch statements

10.16 Case Study :

Create a class Number having the following features:

Attributes

int	first number
int	second number
result	double stores the result of arithmetic operations performed on a and b

Member functions

Number(x, y)	constructor to initialize the values of a and b
add()	stores the sum of a and b in result
sub()	stores difference of a and b in result
mul()	stores product in result
div()	stores a divided by b in result

Test to see if b is 0 and throw an appropriate exception since division by zero is undefined.

Display a menu to the user to perform the above four arithmetic operations.

10.17 Further Reading :

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000
2. Java 2, The Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999
3. Programming with Java, Ed. 2, E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000
4. The Java Tutorial, Ed. 2, 2 volumes, MaryCampione and Kathy Walrath, Addison Wesley Longmans, 1998
5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele & Gilad Bracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000
6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition.

Unit 11

UTILITIES & MULTITHREADING

UNIT STRUCTURE

11.0 Learning Objectives :

After learning this unit, you will be able to :

- Identify utility package
- Describe arrays and hash table
- Explain the thread lifecycle
- Define thread class and runnable interface
- State thread priorities
- Point out synchronisation
- Illustrate deadlock
- Explain suspending, resuming and stopping threads

11.1 Introduction :

Java Utility package is one of the most commonly used packages in the java program. The Utility Package of Java consists of the following components:

- Collections framework
- Legacy collection classes
- Event model

- Date and time facilities
- Internationalisation

Miscellaneous utility classes such as string tokeniser, random–number generator and bit array

In this unit, we will be studying about details of these utility packages as well as about multithreading.

11.2 Comparing Arrays : Java Util :

This section shows you how to determine the given arrays are same or not. The given program illustrates you how to compare arrays according to the content of that in this section, you can see that the given program initializes two arrays and input five numbers from user through the keyboard. And then program checks whether the given taken both arrays are same or not. This comparison operation is performed by using the equals () method of Arrays class.

```
public class ComparingArrays{
   public Arrays.equals():
```

Above method compares two arrays.

Arrays is the class of the *java.util.*;* package. This class and it's methods are used for manipulating arrays.

Here is the code of the program :

```
import java.io.*;
import java.util.*;
classArrayDemo{
static void main(String[] args) throws
   IOException{ int[] array1 = new int[5];
   int[] array2 = new int[5];
BufferedReader br = new BufferedReader(new InputStreamReader
                                                    (System.in));
try{
System.out.println("Enter 5 numbers for the first Array : ")
for(int i = 0; i < array1.length; i++){
array1[i] = Integer.parseInt(br.readLine());
}
System.out.println("Enter 5 numbers for the second Array : ")
for(int i = 0; i < array2.length; i++){
array2[i] = Integer.parseInt(br.readLine());
}
}
catch(NumberFormatException ne){
ne.printStackTrace();
}
boolean check = Arrays.equals(array1, array2);
if(check == false)
System.out.println("Arrays are not same.")
else
System.out.println("Both Arrays are same").
}
}
```

❑ **Check Your Progress – 1 :**

1. Write a note on Arrays.equals().
2. Explain arrays class.

....

....

....

....

....

11.3 Creating a Hash Table : Java Util :

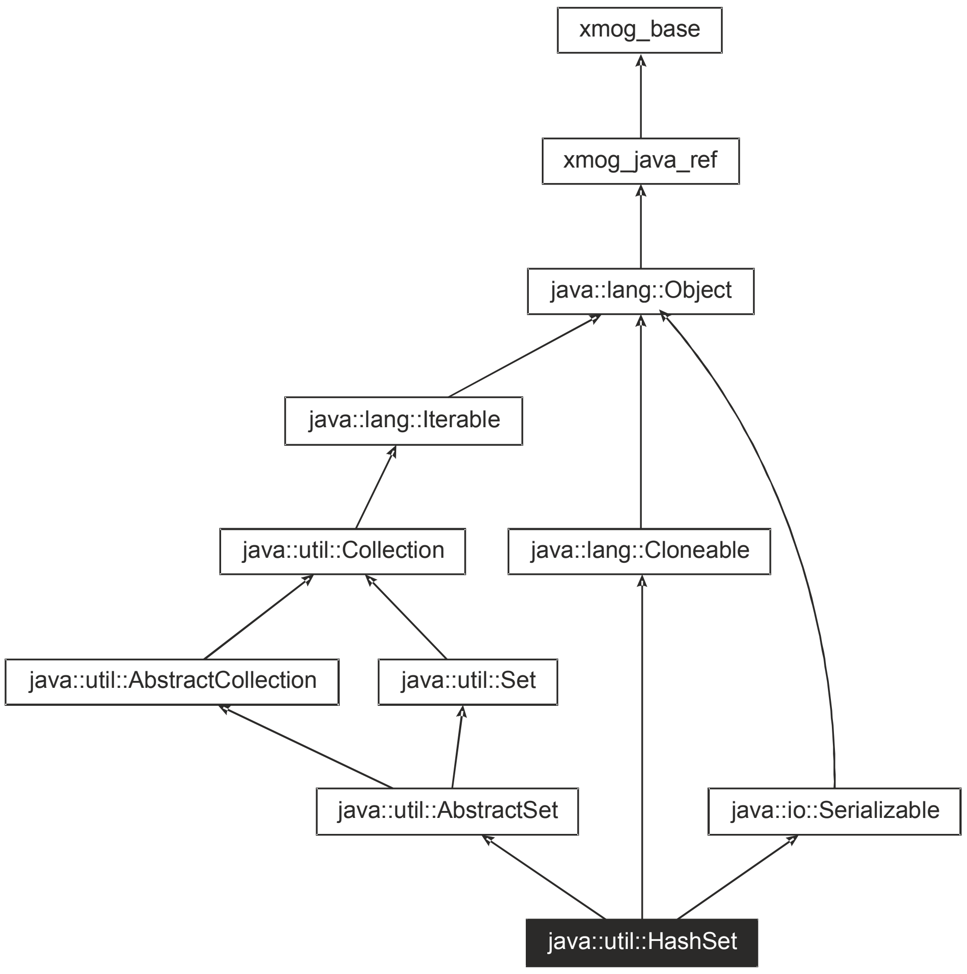


Figure 11.1 Hash Table

This section explains the implementation of the hash table. "What is the hash table and how to create that ? Hash Table holds the records according to the unique key value. It stores the non–contiguous key for several values. Hash Table is created using an algorithm (hashing function) to store the key and value regarding to the key in the hash bucket. If you want to store the value for the new key and if that key is already exists in the hash bucket then the situation known as collision" occurs which is the problem in the hash table i.e. maintained by the hashing function. The drawback is that hash tables require a little bit more memory and that you cannot use the normal list procedures for working with them.

This program simply asks you for the number of entries which have to enter into the hash table and takes one–by–one key and its value as input. It shows all the elements with the separate key.

❖ **Code Description :**

Hashtable<Integer, String> hashTable = new Hashtable<Integer, String>():

Above code creates the instance of the Hashtable class. This code is using the type checking of the elements which will be held by the hash table.

hashTable.put(key, in.readLine()):

Above method puts the values in the hash table regarding to the unique key. This method takes two arguments in which, one is the key and another one is the value for the separate key.

Map<Integer, String> map = new TreeMap<Integer, String> (hashTable):

Above code creates an instance of the TreeMap for the hash table which name is passed through the constructor of the TreeMap class. This code creates the Map with the help of the instance of the TreeMap class. This map has been created in the program for showing several values and it's separate key present in the hash table. This code has used the type checking.

❑ **Check Your Progress – 2 :**

1. What is the drawback for hash tables ?
2. Which method puts the values in hash table ?

....

....

....

....

....

11.4 Multi–Threading :

Figure 11.2 Multithreading

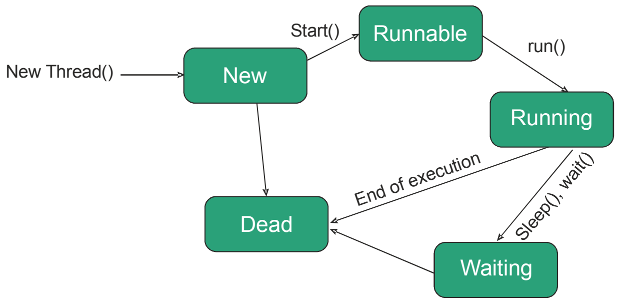


Figure 11.3 Thread Life Cycle

Java provides built–in support for multithreaded programming. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread and each thread defines a separate path of execution.

A multithreading is a specialised form of multitasking. Multitasking threads require less overhead than multitasking processes.

There is another term to be defined related to threads: process: A process consists of the memory space allocated by the operating system that can contain one or more threads. A thread cannot exist on its own; it must be a part of a process. A process remains running until all of the non–daemon threads are done executing.

Multithreading enables you to write very efficient programs that make maximum use of the CPU, because idle time can be kept to a minimum.

❑ **Check Your Progress – 3 :**

1. What do you mean by thread?
2. Explain a process.

....

....

....

....

....

11.5 Thread Life Cycle :

The thread passes many stages in its life cycle including being born, starting, running and dying. Following diagram shows complete life cycle of a thread.

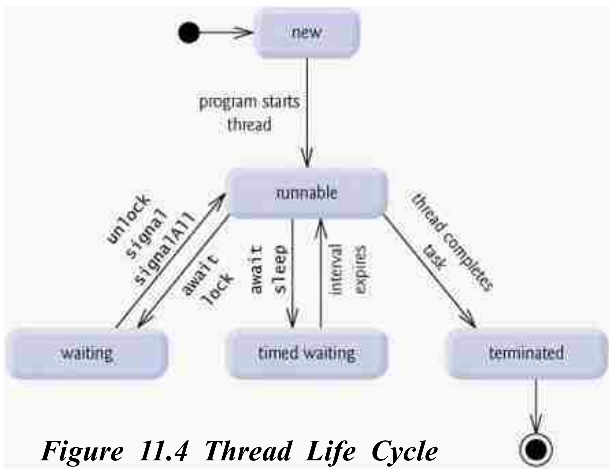


Figure 11.4 Thread Life Cycle

Above mentioned stages are explained here:

- **New** – At this stage the thread is also referred to as a born thread and it is where the life cycle of the thread begins. A thread shall remain in this state until the program starts it.
- **Runnable** – at this stage a thread is executing its task. After it is started a thread becomes runnable.
- **Waiting** – While waiting for another thread to perform its task, a thread tends to transition into the waiting stage. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed waiting** – A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transition back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated** – A runnable thread enters the terminated state when it completes its task or otherwise terminates.

❑ **Check Your Progress – 4 :**

1. Draw a diagram showing complete life cycle of a thread.
2. Explain multithreading.

....

....

....

....

....

11.6 The Thread Class and the Runnable Interface :

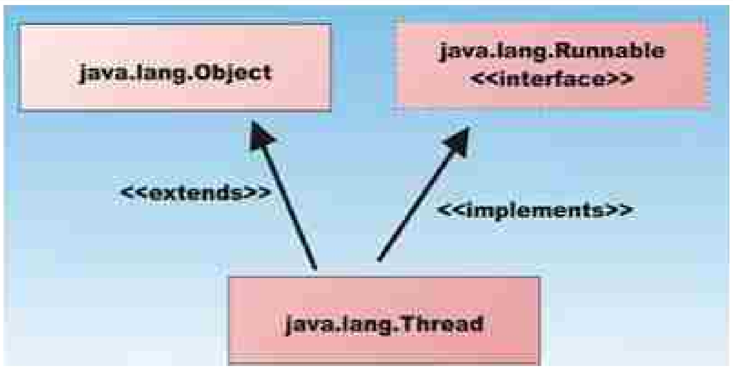


Figure 11.5 Thread Class and Runnable Interface

The easiest way to create a thread is to create a class that implements the Runnable interface.

To implement Runnable, a class need only implement a single method called run ( ), which is declared like this:

public void

run( )

You will define the code that constitutes the new thread inside run() method. It is important to understand that run() can call other methods, use other classes and declare variables, just like the main thread can.

After you create a class that implements Runnable, you will instantiate an object of type Thread from within that class. Thread defines several constructors. The one that we will use is shown here :

Thread(Runnable threadOb, String threadName)

Here *threadOb* is an instance of a class that implements the Runnable interface and the name of the new thread is specified by *threadName*.

After the new thread is created, it will not start running until you call its start ( ) method, which is declared within Thread. The start ( ) method is shown here :

❑ **Check Your Progress – 5 :**

1. What is the easiest way to create a thread ?
2. Explain how to implement runnable interface.

....

....

....

....

....

11.7 Thread Priorities :

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.

Java priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).

Threads with higher priority are more important to a program and should be allocated processor time before lower–priority threads. However, thread priorities cannot guarantee the order in which threads execute and are very much platform dependent.

❖ **Creating a Thread :**

Java defines two ways in which this can be accomplished:

- You can implement the runnable interface.
- You can extend the thread class, itself

❑ **Check Your Progress – 6 :**

1. Define the two ways to create a thread.
2. Give the range of Java priorities?

....

....

....

....

....

11.8 Synchronisation :

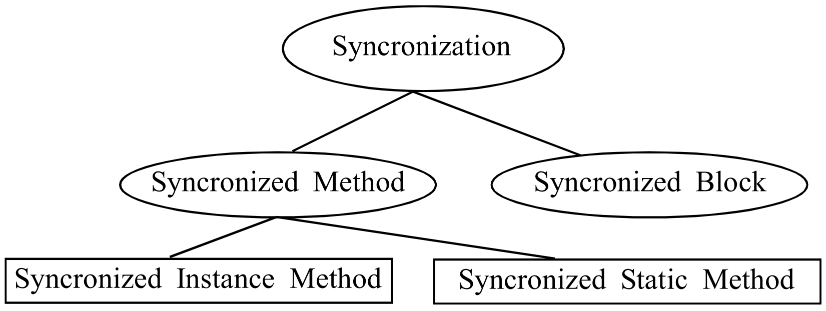


Figure 11.6 synchronisation

When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time.

The process by which this synchronisation is achieved is called thread synchronisation.

The synchronised keyword in Java creates a block of code referred to as a critical section. Every Java object with a critical section of code gets a lock associated with the object. To enter a critical section, a thread needs to obtain the corresponding object's lock.

This is the general form of the synchronised statement:

```
synchronised(object)
{
//statements to be synchronised
}
```

Here, object is a reference to the object being synchronised. A synchronised block ensures that a call to a method that is a member of object occurs only after the current thread has successfully entered object's monitor.

Here is an example, using a synchronised block within the run ( ) method:

```
//The given below program uses a synchronised block
//File Name Call.java
class Callme
{
voidmessage(String msg)
{
System.out.println("[" + msg);
Try
{
Thread. sleep (1000);
}
catch (InterruptedException e)
{
System.out.println ("Interrupted");
}
System.out.println ("]");
}
}
```

//File Name: Caller java

```
class Caller implements Runnable
{
String msg
Callme target;
Thread t;
public Caller (Callme targ, String s)
{
target=targ
msg=s;
t=new thread (this);
t.start ( )
}
//synchronize calls to call ( )
public void run ( )
{
synchronised (target) //synchronised block
{
target.call(msg)
}
}
}
//File Name: Synch.java
Class synch
{
public static void main (String args[])
{
Callme target=new Callme( );
Caller ob1=new Caller (target, "Hello");
Caller ob2=new Caller (target, "Synchronised ");
Caller ob3=new Caller (target,"World");
//Wait for threads to end
try
{
ob1.t.join( );
ob2.t.join( );
ob3.t.join( );
} catch (InterruptedException e)
{
System.out.println ("Interrupted")
}
}
}
```

The above program will produce the given output:

```
[Hello]
[World]
[Synchronised ]
```

❑ **Check Your Progress – 7 :**

1. Write the general form of synchronised statement.
2. What do you mean by critical section ?

....

....

....

....

....

11.9 Deadlock :

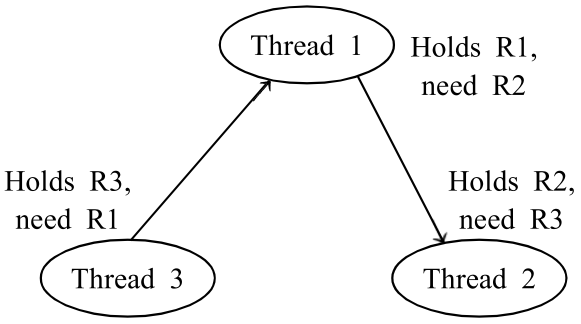


Figure 11.7 Deadlock

A special type of error that you need to avoid that relates specifically to multitasking is deadlock, which occurs when two threads have a circular dependency on a pair of synchronised objects.

For example, suppose one thread enters the monitor on object X and another thread enters the monitor on object Y. If the thread in X tries to call any synchronised method on Y, it will block as expected. However, if the thread in Y, in turn, tries to call any synchronised method on X, the thread waits forever, because to access X, it would have to release its own lock on Y so that the first thread could complete.

Example :

To understand deadlock fully, it is useful to see it in action. The next example creates two classes, A and B, with methods foo ( ) and bar( ), respectively, which pause briefly before trying to call a method in the other class.

The main class, named Deadlock, creates an A and a B instance and then starts a second thread to set up the deadlock condition. The foo ( ) and bar( ) methods use sleep( ) as a way to force the deadlock condition to occur.

```
Class A
{
Synchronised void foo (B b)
{
String name = Thread.currentThread ( ). getName ( );
                              System.out.println (name + "entered a.foo");
try
{
Thread.sleep (2000);
} catch (Exception e)
{
System.out.println ("A Interrupted");
}
System.out.println (name + "trying to call B.last ( )");
b.last ( );
}
Synchronised void last ( )
{
System.out.println ("Inside A.last");
}
}

Class B
{
Synchronised void bar (A a)
{
String name = Thread.currentThread ( ).getName( );
System.out.println (name + "entered B.bar");
try
{
Thread.sleep (1000);
} catch (Exception e)
{
System.out.println ("B Interrupted");
}
System.out.println (name + "trying to call A.last ( )");
a.last ( );
}
Synchronised void last ( )
{
System.out.println ("Inside A.last");
}
}

Class Deadlock implements Runnable
{
A a= new A ( );
B b = new B ( );
Deadlock ( );
{
Thread.currentThread ( ).setName("Main Thread"); Thread t = new Thread
                                        (this, "Racing Thread"); t.start ( );
a.foo (b); //get lock on a in this thread System.out.println
                                                ("Back in main thread");
}
public void run ( )
{
b.bar (a); //get lock on b in other thread
System.out.println ("Back in other thread");
}
public static void main (String args [ ])
{
new Deadlock ( );
}
}
```

The output of the above program will be

```
MainThread entered A.foo
RacingThread entered B.bar
MainThread try to call B.last ( )
RacingThread trying to call
A.last ( )
```

As the program is deadlocked, you need to press CTrl–C to end the program. You can see a full thread and monitor cache dump by pressing Ctrl–Break.

You will see that Racing Thread owns the monitor on b, while it is waiting for the monitor on a. At the same time, Main Thread owns a and is waiting to get b. This program will never complete.

As this example illustrates, if your multithreaded program locks up occasionally, deadlock is one of the first condition that you should check for.

Ordering Locks

A common threading trick to avoid the deadlock is to order the locks. By ordering the locks, it gives threads a specific order to obtain multiple locks.

❑ **Check Your Progress – 8 :**

1. What is the trick to avoid deadlock?
2. What should be checked if the multithreaded program is locked occasionally ?

....

....

....

....

....

11.10 Suspending, Resuming and Stopping Threads :

While the suspend ( ), resume ( ) and stop( ) methods defined by Thread class seem to be a perfectly reasonable and convenient approach to managing the execution of threads, they must not be used for new Java programs and obsolete in newer versions of Java.

The following example illustrates how the wait ( ) and notify ( ) methods that are inherited from Object can be used to control the execution of a thread.

This example is similar to the program in the previous section. However, the deprecated method calls have been removed. Let us consider the operation of this program.

The NewThread class contains a boolean instance variable named suspendFlag, which is used to control the execution of the thread. It is initialised to false by the constructor.

The run( ) method contains a synchronised statement block that checks suspendFlag. If that variable is true, the wait( ) method is invoked to suspend the execution of the thread. The mysuspend( ) method sets suspendFlag to true. The myresume( ) method sets suspendFlag to false and invokes notify( ) to wake up the thread. Finally, the main( ) method has been modified to invoke the mysuspend( ) and myresume( ) methods.

❑ **Check Your Progress – 9 :**

1. Write a multithreaded program to create threads and use suspend and resume methods to transfer control.
2. Explain the execution of suspend, resume and stop threads.

....

....

....

....

....

3. Java ________ package is one of the most commonly used packages in the java program.

   (A) Net (B) IO (C) Utility (D) Lang

4. ________ is not the components of JAVA Utility

   (A) Collections framework (B) Legacy collection classes

   (C) Event model (D) Illustrate deadlock

5. ________ at this stage a thread is executing its task. After it is started a thread becomes ________.

   (A) New (B) Runnable (C) Waiting (D) Terminated

6. To implement Runnable, a class need only implement a single method called ________ ( ),

   (A) Start (B) Run (C) Terminate (D) Void

7. ________ A runnable thread enters the terminated state when it completes its task or otherwise ________.

   (A) New (B) Runnable (C) Waiting (D) Terminated

11.11 Let Us Sum Up :

This section shows you how to determine the given arrays are same or not. The given program illustrates you how to compare arrays according to the content of that.in this section, you can see that the given program initializes two arrays and input five numbers from user through the keyboard. And then the program checks whether the given taken both arrays are same or not. This comparison operation is performed by using the equals () method of Arrays class. This section explains the implementation of the hash table. What is the hash table and how to create that? Hash Table holds the records according to the unique key value. It stores the non–contiguous key for several values. Hash Table is created using an algorithm (hashing function) to store the key and value regarding to the key in the hash bucket. If you want to store the value for the new key and if that key is already exists in the hash bucket then the situation known as collision. It occurs when there is the problem in the hash table maintained by the hashing function.

Java provides built–in support for multithreaded programming. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread and each thread defines a separate path of execution.

A multithreading is a Specialized form of multitasking. Multitasking threads require less overhead than multitasking processes. There is another term to be defined related to threads: process: A process consists of the memory space allocated by the operating system that can contain one or more threads. A thread cannot exist on its own; it must be a part of a process. A process remains running until all of the non–daemon threads are done executing.

There is also understanding related to Thread life cycle. A thread goes through various stages in its life cycle. For example, a thread is born, started, runs and then dies. Following diagram shows complete life cycle of a thread.

There is a mention about Thread Priority, every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.

Java priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).

Threads with higher priority are more important to a program and should be allocated processor time before lower–priority threads. However, thread priorities cannot guarantee the order in which threads execute and are very much platform dependent. There is a learning related to When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this synchronization is achieved is called thread synchronization.

The synchronized keyword in Java creates a block of code referred to as a critical section. Every Java object with a critical section of code gets a lock associated with the object. To enter a critical section, a thread needs to obtain the corresponding object's lock.

There is understanding related to a Dead lock, a special type of error that you need to avoid that relates specifically to multitasking is deadlock, which occurs when two threads have a circular dependency on a pair of synchronised objects.

About Suspending, Resuming and Stopping Threads

Further there is a understanding about While the suspend ( ), resume ( ) and stop( ) methods defined by Thread class seem to be a perfectly reasonable and convenient approach to managing the execution of threads, they must not be used for new Java programs and obsolete in newer versions of Java.

11.12 Suggested Answer for Check Your Progress :

- **Check Your Progress 1 :**
  See Section 11.2
- **Check Your Progress 2 :**
  See Section 11.3
- **Check Your Progress 3 :**
  See Section 11.4
- **Check Your Progress 4 :**
  See Section 11.5
- **Check Your Progress 5 :**
  See Section 11.6
- **Check Your Progress 6 :**
  See Section 11.7
- **Check Your Progress 7 :**
  See Section 11.8
- **Check Your Progress 8 :**
  See Section 11.9

❑ **Check Your Progress 9 :**

1 : See Section 11.10 **2 :** See Section 11.10

3 : C **4 :** D **5 :** B **6 :** B **7 :** D

11.13 Glossary :

1. **Collision** – If you want to store the value for the new key and if that key is already exists in the hash bucket then the situation known as collision occurs which is the problem in the hash table maintained by the hashing function.
2. **Multithreading** – A multithreading is a Specialized form of multitasking. Multitasking threads require less overhead than multitasking processes.
3. **Synchronized keyword** – The synchronized keyword in Java creates a block of code referred to as a critical section. Every Java object with a critical section of code gets a lock associated with the object.

11.14 Assignment :

Write a program to create multiple threads

11.15 Activities :

1. Describe the important methods available in thread class.
2. Write a program to remove deadlock

11.16 Case Study :

Every program has at least one thread. Programs without multithreading executes sequentially. That is, after executing one instruction the next instruction in sequence is executed. If a function is called then until the completion of the function the next instruction is not executed. Similarly if there is a loop then instruction safer loop only gets executed when the loop gets completed. Consider the following java program having three loops.

11.17 Further Reading :

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000
2. Java 2, The Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999
3. Programming with Java, Ed. 2,E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000
4. The Java Tutorial, Ed. 2, 2 volumes, Mary Campione and Kathy Walrath, Addison Wesley Longmans, 1998
5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele & Gilad Bracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000
6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition, 2000

BLOCK SUMMARY :

The Unit No. 9 of this Block we have understood Inheritance is one of the cornerstones of OOP because it allows for the creation of hierarchical classifications. Using inheritance, you can create a general class that defines traits common to a set of related objects, that is, objects with common attributes and behaviors. The various needs of inheritance are 1. Closer to Real–World, 2. Code Reusability ad 3. Transitive Nature

The study of Generalisation/ Specialisation has made us understand Syntax, Create a superclass then Create a subclass by extending class A and also the subclass has access to all public members of its superclass. Then further studied that Polymorphism allows one interface to be used for a set of actions i.e. one name may refer to different functionality. Polymorphism allows an object to accept different requests of a client (it then properly interprets the request like choosing appropriate method) and responds according to the current state of the runtime system, all without bothering the user. There are two types of polymorphism, 1. Compile–time polymorphism, 2. Runtime Polymorphism.

We understood Method Overriding, if a class inherits a method from its super class, then there is a chance to override the method provided that it is not marked final. Overriding means redefining a method in an inheritance hierarchy. In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass then the method in the subclass is said to override the method in the superclass. We have also understood in detail all Rules for method overriding. There is also good understanding about the final keyword has mainly three uses 1. Creating constants, 2.Preventing method overriding, and 3. Preventing inheritance

Unit No. 10 Deals with an exception is a problem that arises during the execution of a program. It is a runtime error. In some of the languages, which do not support exception handling, errors must be checked and handled manually typically through the use of error codes and so on. This approach is quite cumbersome and troublesome.

Then there is understanding relating to the predefined Exception, Java provides several predefined Exception classes in the package java.lang. To understand how exception handling works in Java, you need to understand these categories of exceptions :

1. Checked exceptions
2. Unchecked exceptions
3. Errors

"When the exceptions are not caught in a try/catch block, then what you often see in practice is Java prints the exception stack trace and then terminates your program. Java actually handles uncaught exceptions according to the thread in which they occur. When an uncaught exception occurs in a particular thread, Java looks for what is called an uncaught exception handler, actually an implementation of the interface Uncaught Exception Handler. The latter interface has a method handle Exception (), which the implementer overrides to take appropriate action, such as printing the stack trace to the console."

We have understood that Java Exception handling is managed using five keywords – a. try, b. catch, c. throw, d. throws and finally e. finally.

Catching Exceptions : A method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code

User defined exceptions : You can create your own exceptions in Java. Keep the following points in mind while writing your own exception classes:

- All exceptions must be a child of Throwable.
- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
- If you want to write a runtime exception, you need to extend the RuntimeException class.

Using Throws/throw Keywords: This means that there should be a code somewhere in the program that could catch the exception. We use throw statement to throw an exception or simply use the throw keyword with an object reference to throw an exception. Know after that we understood. The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred.

Using a finally block allows you to run any cleanup–type statements that you want to execute, no matter what happens in the protected code.

We have also understood about Nested Try Statement in Java we can have nested try and catch blocks. It means that, a try statement can be inside the block of another try. If an inner try statement does not have a matching catch statement for a particular exception, the control is transferred to the next try statement's catch handlers that are expected for a matching catch statement. This continues until one of the catch statements succeeds, or until the entire nested try statements are done in. If no one catch statements match, then the Java run–time system will handle the exception

BLOCK ASSIGNMENT :

❖ **Short Questions :**

1. What are checked and unchecked exceptions ?
2. Why super keyword used ?
3. How to prevent deadlocks between threads ?
4. If a class is final class, can we have extended class? State true/false.
5. What is static polymorphism and dynamic polymorphism ?

❖ **Long Questions :**

1. Life Cycle of Thread
2. Difference between Overloading and Overriding methods with an example ?
3. Create a class USERTRAIL with following specifications.

   val1, val2 type int

   Methods:

   boolean show () will check if val1 and val2 are greater or less than Zero.

   Have constructor which will val1, val2 and check whether if it is less than 0 then raise a custom Exception (name : Illegal value exception.)

❖ **Enrolment No. :** ☐

1. How many hours did you need for studying the units ?

Unit No.	9	10	11
No. of Hrs.			

2. Please give your reactions to the following items based on your reading of the block :

Items	Excellent	Very Good	Good	Poor	Give specific example if any
Presentation Quality	☐	☐	☐	☐	________ ________
Language and Style	☐	☐	☐	☐	________ ________
Illustration used (Diagram, tables etc)	☐	☐	☐	☐	________ ________
Conceptual Clarity	☐	☐	☐	☐	________ ________
Check your progress Quest	☐	☐	☐	☐	________ ________
Feed back to CYP Question	☐	☐	☐	☐	________ ________

3. Any other Comments

....

....

....

....

....

....

....

....

Dr. Babasaheb Ambedkar Open University Ahmedabad

BCAR-204/ DCAR-204

Object Oriented Concepts & Programming–1 (Core Java)

BLOCK 4 : ABSTRACT WINDOW TOOLKIT AND WORKING WITH FILES

ABSTRACT WINDOW TOOLKIT AND WORKING WITH FILES

Block Introduction :

The learning about window toolkits has great details relating to the Window fundamentals, Explanation related working with Applet, Applet's graphics and controls, State layout managers and event handling, Describe adapter classes, inner classes, anonymous inner classes, Discuss applet fundamentals and applet lifecycle, Define i/o streams, Describe reading console input and writing console output, Explain reading and writing files and Discuss serialization. This is also important as long as bringing confidence level amongst learner about further Programming part.

Block Objectives :

After learning this block, you will be able to :

- Explain working with Applet and Applet's Graphics
- Explain working with graphics and controls
- State layout managers and event handling
- Describe adapter classes, inner classes, anonymous inner classes
- Discuss applet fundamentals and applet lifecycle.
- Define i/o streams
- Describe reading console input and writing console output
- Explain reading and writing files
- Discuss serialization

Block Structure :

Unit 12

APPLET

UNIT STRUCTURE

12.0 Learning Objectives :

After learning this unit, you will be able to understand the use of :

- Difference between Applet and Application
- Applet Life Cycle
- Applet Tag
- How to creating Applet
- Reading parameters into Applet
- Implementation of Background colour in Applet
- Implementation of Font colour in Applet

12.1 Introduction :

Java applet is a special kind of program that execute in web browser. Java applet program is downloaded from internet and run on web browser. Java applet is classically wrapped inside a web page and execute in the context of web browser. To Wrapped up it used Hypertext Markup language (HTML). The Java applet does not have any control of local computer.

Downloading an applet code from the internet and executing it on our local computer is sometimes dangerous, as that applet any have some spurious code. Therefore, the java developers have introduced some security restrictions regarding the downloading and usage of the applets. For example, applets are not allowed to read or write to our local disk. This restriction is necessary, as an applet may accidentally or wantonly destroy any data stored in the local machine. Applets are split into small packets named bytecodes and travel across the network. These byte codes are reassembled by the Java Virtual Machine (JVM) in the receiving machine and executed by the web browser. The Applet does not executed directly.

The browser that executes an applet is generally known as applet container. The JDK includes the appletviewer, for testing applets as we develop them and before we embed them in the browser.

The applet is a subclass of the *java.applet.Applet* class. The Applet class provide an interface between the applet and web browser environment. The Applet class hierarchy is shown below:

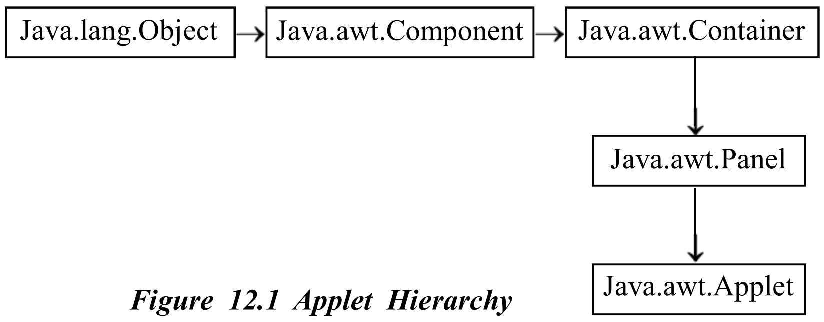


Figure 12.1 Applet Hierarchy

The following program show simple applet :

```
import java.awt.*;
import java.applet.*;

public class Appletdemo extends Applet
{
        public void paint(Graphics g)
        {
                g.drawString("The First Applet", 65,35);
        }
}
```

Here you may notice that java program does not contain main() method and System.out.println() method. How to save, compile and execute the applet program will learn latter on in this unit. Here we need to understand that Applets use only graphics methods for show any output which we called window based output. For showing window based output java required graphics supports. To generate graphics applet required Java's Abstract Window Toolkit (AWT). Therefore to create applet program we required *java.applet* and *java.awt* packages to be imported.

12.2 Difference between Applet and Application :

Java application runs by JIT under JVM. Java application can be CUI or GUI interface. Applet can be run on either applet viewer tool or using web browser. For the security purpose applet does not have rights to access the local machine resource. Application can be access the local machine's resource. Applet execute at remote server and run on local machine. Application execute and run on local machine.

Table 12.1 : Applet v/s Application

Applet	Application
Every Applets must have paint() method. Paint method is used to paint the applet.	Every Application have main() method, which is starting point of Application.
Applet require web browser for their execution.	Application can be directly executed.
Applet can be run using appletviewer tool.	Java interpreters (JIT) run the application.
An applets written by any developer in the world and may be dynamically download from a web server and execute on a client PC.	We require JVM to run java application.

12.3 Applet Life Cycle :

Let us learn about Applet Lifecycle. Applet runs in the browser and their lifecycle methods are called by JVM after it is loaded and destroyed. You can see here the lifecycle methods of an Applet :

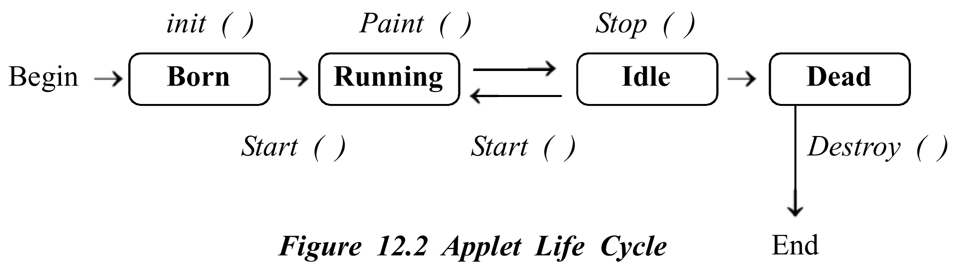


Figure 12.2 Applet Life Cycle

init(): method of this kind is called to initialised an applet.

start(): method of this kind is called after the initialization of the applet.

stop(): This method would be called multiple times in the life cycle of an Applet.

destroy(): This method is called once in the life cycle of the applet when applet is destroyed.

init () method: Now we need to understand that the life cycle of an applet is going to begin when there a time the applet is first loaded into the browser and then called the init() method. The init() method is called only one time in the life cycle on an applet. The init() method is mostly called to read the PARAM tag in the html file. The init () method retrieves the passed parameter through the PARAM tag of html file using get Parameter() method. The various initialization such as initialization of variables and the objects like image, sound file are loaded in the init () method thereafter the initialization of the init () method then user can interact with the Applet and generally applet contains the init() method.

```
public void init()
{
        ......
        ......
}
```

Start () method: In this method an applet is called after the initialization method init(). Hence this method may be called multiple times when the Applet needs to be started or restarted. To understand this we take an example, in case of the user wants to return to the Applet, then in this situation the start Method() of an Applet will be called by the web browser and the user will be back on the applet. In the start method user can interact within the applet.

```
public void start()
{
        ……
        ……
}
```

The paint() method is called by the applet container after start() method. This method is used whenever applet has to perform some output operations on the screen. Typical actions performed here involve drawing with the graphics object g that is passed to the paint method by the applet container.

```
public void paint(Graphics g)
{
        ……
        ……
}
```

Stop () method: this stop () method can be called many times in the life cycle of applet like the start () method or otherwise should be called at least one time. There is only minor difference between the start() method and stop () method. Take for example, the stop() method is called by the web browser on that time or occasion when the user leaves one applet to go to another applet and the start() method is called on that time, when the user wants to go back into the first program or Applet.

```
public void stop()
{
        ……
        ……
}
```

destroy() method: this destroy() method is called only one(once) time in the life cycle of Applet like init() method. This method is going to be called only on that time when the browser needs to Shut down.

```
public void destory()
{
        ……
        ……
}
```

12.4 Creating an Applet :

We already see the applet program in unit 1.1. Now its time to discuss the program in detail. For creating and executing the applet program we need to follow the bellow steps :

1. writing the applet code and save it in .java file
2. compiling the java file and generate .class file
3. writing the HTML code and save it as .html file
4. invoking the appletviewer tool and execute the command "appletviewer Appletdemo.html"

The following program show simple applet :

```
import java.awt.*;
import java.applet.*;

public class Appletdemo extends Applet
{
        public void paint(Graphics g)
        {
                g.drawString("The First Applet", 65,35);
        }
}
```

Save this program with Appletdemo.java and compile it on command prompt as follow :

```
javac Appletdemo.java
```

Now create other file which contains HTML tag. This is below :

```
<html>
        <head>
        </head>

        <body>
                <applet code = "FirstApplet.class" width= 500 height=400>
                </applet>
        </body>
</html>
```

Save the above code with named Appletdemo.html

Now for Running an Applet write following on command prompt:

```
C:> appletviewer Appletdemo.html
```

In the above program, applet window has a width 500 and height 400 pixel. The Strign "The First Applet" is displayed at the x,y co–ordinate 65, 35 of the display area.

Here note that for the purpose of displaying the figures on the screen, the upper left corner of the screen is always taken as the origin(0,0). While the x–coordinate increases horizontally to the right, the y–coordinate increases vertically downwards, as shown below.

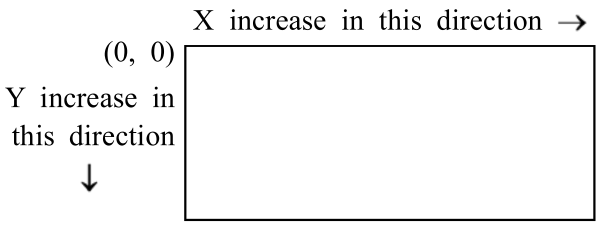
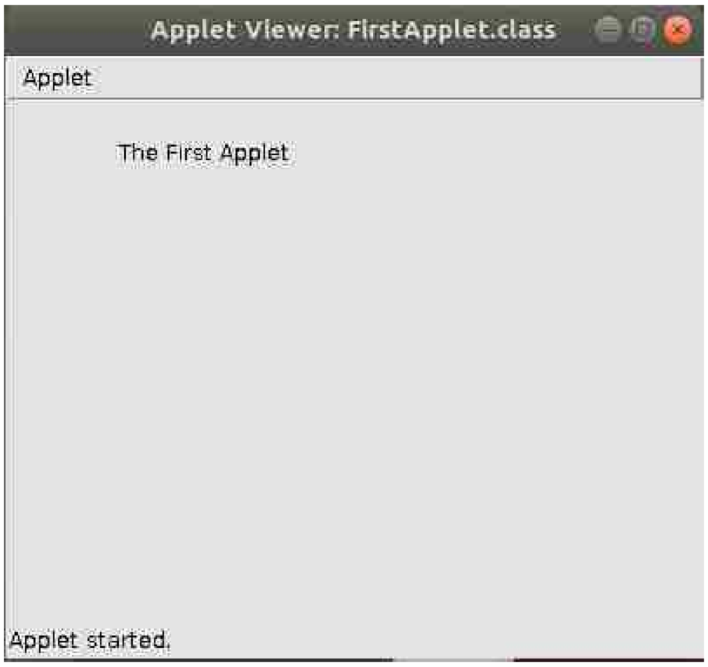

X increase in this direction →
(0, 0)
Y increase in this direction ↓

Figure 12.3 Applet Starting Point

Instead of creating two programs, an applet program and another html file to execute the applet, the applet element of html file can be embedded within the applet program itself as a comment. By this way, the html and applet program can be combined and executed as a single program.

Following program shows how to create Applet. In this program HTML code is write in .java file.

```
import java.awt.*;
import java.applet.*;

/*
<applet code = "FirstApplet.class" width= 400 height=300>
</applet>
*/

public class FirstApplet extends Applet
{
        public void paint(Graphics g)
        {
                g.drawString("The First Applet", 65,35);
        }
}
```



```
The First Applet
```

```
Applet started.
```

Figure 12.4 Output of Program

To invoke above program, write the following on command prompt

appletviewer FirstApplet.java

Now we need to discuss some important methods of Applet. The bellow table 12.2 gives a brief description of methods available under Applet Class:

Table 12.2 : Methods defined in Applet Class

Method	Description
String getAppletInfo()	Returns the details of applet in String
String getParameter(String name)	Returns parameter 'name' in string
String[][] getParameterInfo()	Returns a string table that is defined in the applet
URL getCodeBase()	Returns the URL associated with the applet
URL getDocumentBase()	Returns the URL of the HTML page that contains the applet.
boolean isActive()	Returns true if applet started and returns false if applet is stopped.
void resize(int width, int height)	Resize the window as per 'width' and 'height'
void showStatus(String s)	Show the status 's' in window browser

❑ **Check Your Progress – 1 :**

1. Explain Applet Life Cycle.

....

....

....

....

....

12.5 Applet Tag :

The Applet tag is used to start an applet from both an HTML document and from an applet viewer.

Syntax of Applet Tag :

```
<Applet
[CODEBASE = codebase URL]
CODE = appletFile
[ALT = alternate Text]
[NAME = appletInstanceName]
WIDTH = pixels
HEIGHT = pixels
[ALIGN = ALIGNMENT]
[VSPACE = pixels]
[HSPACE = pixels]
```

```
>
[<PARAM NAME = attribute 1 VALUE = attribute Vale1>]
[<PARAM NAME = attribute 2 VALUE = attribute Vale2>]
………
</Applet>
```

CODEBASE

If the applet resides in the same directory in which the html file resides, this entry may be omitted. Otherwise, this entry is used to specify the URL of the directory in which the applet resides.

CODE

To specify the name of the class file that has been obtained by compiling the applet program. This attribute is compulsory.

ALT

This entry is optional. Certain web browsers are non–java enabled browser. They cannot recognize and execute the applet code. Such browsers will display this alternate text in the place of the applet's output.

NAME

This entry is optional. This entry specifies a name for the applet. Other applets on the same page can refer to this applet with the help of this name.

WIDTH and HEIGHT

To specify the width and height of the applet output frame. This frame is part of the web page.

ALIGN

This entry is optional. This entry specifies the alignment, according to which the applet's output will appear. Possible values are LEFT, RIGHT, TOP, TEXTTOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM, ABSBOTTOM.

VSPACE

This entry is optional. This entry specifies the amount of vertical blank space that the web page should have surrounding the applet's output frame.

HSPACE

This entry is optional. This entry specifies the amount of horizontal blank space that the web page should have surrounding the applet's output frame.

PARAM

The PARAM is used to specify applet specific arguments in an HTML page.

❑ **Check Your Progress – 2 :**

1. Explain steps for creating applet.
2. Explain the various method of applet class.

....

....

....

....

....

12.6 Reading Parameters into Applet :

Applet can also read the data using parameters. To read the data applet tag use "PARAM" attribute. By using the attribute applet bring data into applet. The getParameter( ) method returns the value of parameter in String format. For other than String type data we need to explicit convert string type to concern type.

Following applet retrieves String values from PARAM tag which is associated with HTML tag.

```
import java.awt.*;
import java.applet.*;

/*
<applet code = "ParameterDemo.class" width= 400 height=300>
<param name = "uniname" value = "BAOU">
</applet>
*/

public class ParameterDemo extends Applet
{
    public void paint(Graphics g)
    {
        String s = getParameter("uniname");
        g.drawString(s, 100,100);
    }
}
```

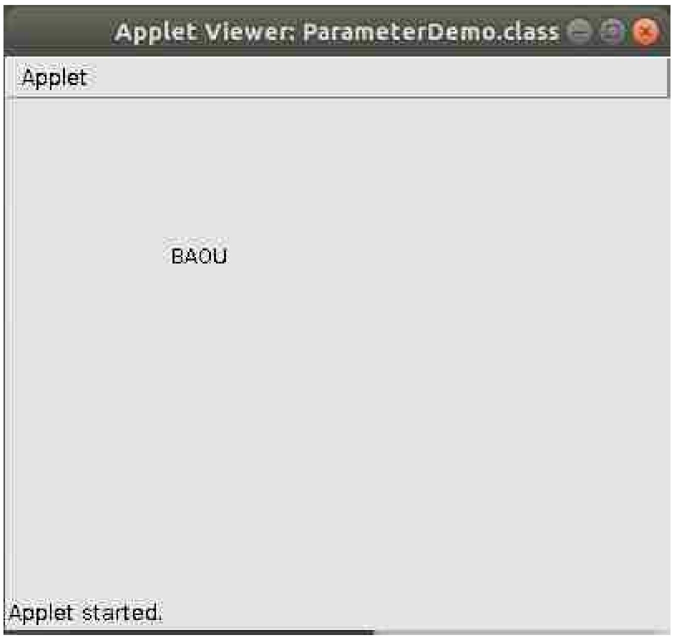
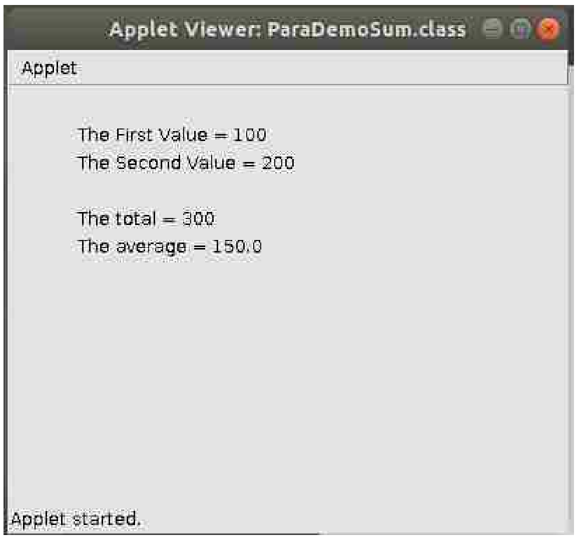


Figure 12.5 Output of Program

Following applet retrieves int values from PARAM tag which is associated with HTML tag.

```
import java.awt.*;
import java.applet.*;

/*
<applet code = "DemoSum.class" width= 400 height=300>
<param name = "v1" value = "100">
<param name = "v2" value = "200">
</applet>
*/

public class ParaDemoSum extends Applet
{
        public void paint(Graphics g)
        {
                String s1 = getParameter("v1");
                String s2 = getParameter("v2");

                g.drawString("The First Value = " + s1 , 50,40);
                g.drawString("The Second Value = " + s2 , 50,60);

                int  n1 = Integer.parseInt(s1);
                int  n2 = Integer.parseInt(s3);

                int total = n1 + n2;
                float average = total / 2;

                g.drawString("The total = " + total , 50,100);
                g.drawString("The average = " + average , 50,120);

        }
}
```



```
The First Value = 100
The Second Value = 200

The total = 300
The average = 150.0
```

Figure 12.6 Output of Program

❑ **Check Your Progress – 3 :**

1. Explain Applet Tag in detail.

....

....

....

....

....

12.7 Implementation of Background Colour in Applet :

We can set the background colour of applet using the value defined in 'Color' class. The Color values defined in Color class as follow :

Table 12.3 : Color Values Defined in Color Class

Color.black	**Color.lightGray**	**Color.red**
Color.darkGray	**Color.blue**	**Color.gray**
Color.magenta	**Color.white**	**Color.cyan**
Color.green	**Color.pink**	**Color.yellow**

We can also set the foreground colour of applet using the values defined in 'Color' class. To set or get the background colour of applet, the method used for the same is define in 'Component' which is mention as follow:

Table 12.4 : Methods for set / get Background colour

Method	**Description**
void setBackground(Color value)	'value' colour set as background in applet.
Color getBackground()	It returns the color name which set as a background of applet.

Color is achieved through three primary colors – red, green, blue. It's range is 0 to 255.

The Syntax of the constructor of color

```
Color(int red, int green, int blue);
Color(int rgbValue);
```

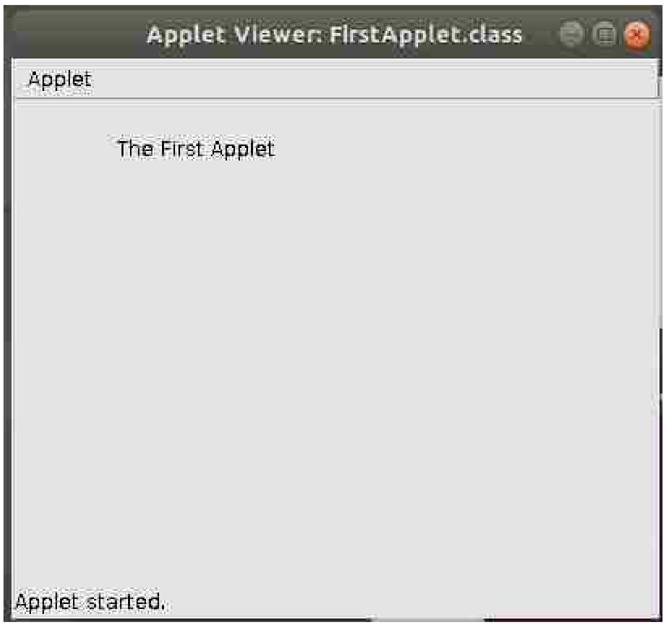
For Example:

```
Color c = new Color(255,100,100);
g.setColor(c);
```

Following applet implements background color of applet.

```
import java.awt.*;
import java.applet.*;

/*
<applet code = "FirstApplet.class" width= 400 height=300>
</applet>
*/
```

```
public class FirstApplet extends Applet
{
        public void paint(Graphics g)
        {
                setBackground(Color.black);
                g.drawString("First Applet", 65,35);
        }
}
```



```
The First Applet
```

```
Applet started.
```

Figure 12.7 Output of Program

12.8 Implementation of Font colour in Applet :

We can also set the foreground colour of applet using the values defined in 'Color' class. The Color values defined in previous section. To set or get the foreground colour of applet, the method used for the same is define in 'Component' which is mention as follow:

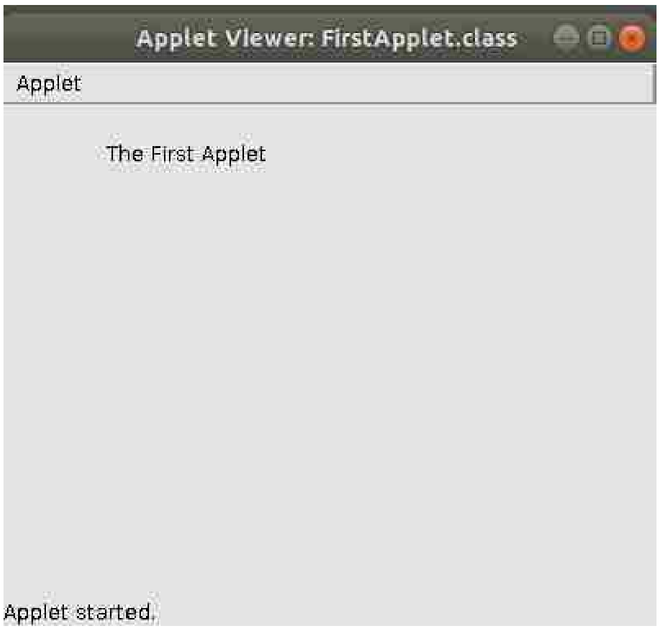
Table 12.5 : Methods for set / get Foreground colour

Method	Description
Void setForeground(Color value)	'value' colour set as foreground in applet.
Color getForeground()	It returns the color name which set as a foreground of applet.

Following applet implements foreground color of applet.

```
import java.awt.*;
import java.applet.*;

/*
<applet code = "FirstApplet.class" width= 400 height=300>
</applet>
```

```
*/

public class FirstApplet extends Applet
{
        public void paint(Graphics g)
        {
                setBackground(Color.black);
                setForeground(Color.white);
                g.drawString("First Applet", 65,35);
        }
}
```



```
The First Applet
```

Figure 12.8 Output of Program

❑ **Check Your Progress – 4 :**

1. Explain parameter reading in to applet.

....

....

....

....

....

2. Applet is subcalss of ________.

   (A) java.applet.Applet (B) java.applet.AWT

   (C) java.applet.IO (D) java.applet.Thread

3. AWT stand for ________.

   (A) Abstract Window Tool (B) ALL Window Toolkit

   (C) Abstract Window Toolkit (D) Abstract Window Text

4. The first method of applet life cycle is ________.

   (A) On( ) (B) paint ( ) (C) start( ) (D) init( )

5. The ________ method is called by the applet container after start() method.

   (A) stop( ) (B) paint() (C) start( ) (D) init( )

6. ________ method is used whenever applet has to perform some output operations on the screen.

   (A) stop( ) (B) paint() (C) start( ) (D) init( )

7. ________ method is called only one(once) time in the life cycle of Applet like init() method. .

   (A) stop( ) (B) paint() (C) start( ) (D) destory( )

8. To execute the applet ________ tool is used.

   (A) java (B) applet (C) appletviewer (D) paint

9. ________ method of applet class returns the details of applet in String.

   (A) getAppletInfo( ) (B) info( )

   (C) getApplet( ) (D) Detail( )

10. The ________ tag is used to start an applet from both an HTML document and from an applet viewer.

    (A) Head (B) Body (C) Applet (D) Java

11. The ________ method returns the value of parameter in String format.

    (A) param( ) (B) getvalue( ) (C) applet( ) (D) getParameter( )

12.9 Let Us Sum Up :

In this Unit we have learned about applet which is defined by under 'Java.applet.Applet' package. We also discuss the different between Java application and Java applet. Java applet does not have any rights to access resources of local machine. Applet always called from remote machine and run on local machine's web browser. To execute applet we require 'appletviewer' tool. Java does not have main( ) method and system.out.println( ) method. Applet is graphical interface which draw using paint( ) method.

Applet has its own life cycle that called applet life cycle. Applet transfer from one stage to other stage during its life cycle, the stages are 'Born', 'Running', 'Idle', 'Dead'. Applet need to called various method to transfer from one stage to other stage like init( ), start( ), stop( ), destroy( ) methods.

To execute applet we required HTML code containing by .html file. The appletviewer tool use this HTML code to execute applet in web browser. We can pass data into applet using "PARAM" attribute. The getParameter( ) method is used to get the value of parameter into applet. At the end we see how to change the background colour and foreground colour of applet using 'Color' class.

Applet has some advantage and disadvantage. The advantages are, applet can work on all installed versions of java. Applet can work without any security approval from the user. Applet is supported by most web browsers. Applet will cache in most web browsers so will be quick to load when returning to a web page. Applet can move the work from the server to the client, making a web solution more scalable with the number of users/clients. The disadvantage are, Applet requires the java plug–in which is not available by default on all web browsers. Applet cannot start up until JVM is running. If applet is uncached, it must be downloaded and this takes time.

12.10 Suggested Answer for Check Your Progress :

❑ **Check Your Progress 1 :**

See Section 12.3

❑ **Check Your Progress 2 :**

See Section 12.4

❑ **Check Your Progress 3 :**

See Section 12.5

❑ **Check Your Progress 4 :**

1 : See Section 12.6		**2 :** A	**3 :** C
4 : D	**5 :** B	**6 :** B	**7 :** D
8 : C	**9 :** A	**10 :** C	**11 :** D

12.11 Glossary :

1. **Applet** – Applet runs on web browser and does not access the local machine resource.
2. **Applet Life Cycle** – Applet has four stages, 'Born', 'Running', 'Idle', 'Dead' under applet life cycle.
3. **Applet Tag** – The Applet tag is used to start an applet from both an HTML document and from an applet viewer.
4. **Reading data into Applet** – The PARAM attribute of applet tag is used to pass the data in to applet.
5. **Color class** – Color class is used to set background and foreground color of applet.

12.12 Assignment :

1. Write a note on Applet life cycle.
2. Write a note on Applet Tag.
3. Explain how to pass the data in to applet

12.13 Activities :

1. Write a program to show the different stage of Applet life cycle.
2. Write a program that set the yellow background colour of applet.
3. Write a program that set the blue background colour and yellow foreground colour of applet.

12.14 Case Study :

1. Prepare the Chart of four stage of applet life cycle.

12.15 Further Reading :

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000.
2. Java 2, the Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999.
3. Programming with Java, Ed. 2, E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000.
4. The Java Tutorial, Ed. 2, 2 volumes, Mary Campione and Kathy Walrath, Addison Wesley Longmans, 1998.
5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele & Gilad Bracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000.
6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition, 2000

Unit 13

APPLET GRAPHICS

UNIT STRUCTURE

13.0 Learning Objectives :

After learning this unit, you will be able to understand the use of :

- Applet Graphics
- Can draw line in Applet
- Can Oval and Circle line in Applet
- Can draw Rectangle and Square in Applet
- Can draw Arcs in Applet
- Can draw Polygons and Polyline in Applet
- Delegation Event Model
- Basics of Form Design and Can draw Form using Text field, Text Area and Button.

13.1 Introduction :

Java provide various kind of tools for designing graphics and user interface. Such tools are available under the Abstract Window Toolkit (AWT) package. The AWT package contains large number of classes for the graphics and user interface. The AWT provides controls related to form designing such as Textfield, TextArea, Button, Radio Button etc. Here, in this unit we discuss the various graphics method that generate the shapes. Generally for the graphics shapes we need to create the window and then graphics object create the shapes. To create the window here we use applet. For create the shapes first we creating applet window. The applet window is closable, it means once we click on close icon (X) on the applet window, the applet window can be closed. Other way to creating the window is using Frame class.

Java provides the graphics tools under 'java.awt.Graphics' class. We know that applet is obtain from 'java.applet.Applet' class. For drawing the graphics shape we need to pass the Graphics object to the paint ( ) method.

13.2 Drawing Line :

To draw the line in applet we need to use following method.

Syntax :

```
public void drawLine(int x1, int y1, int x2, int y2);
```

This method draws the line with current color form (x1,y1) to (x2,y2) points.

Following program draws Line on applet :

```
import java.awt.Graphics;
import java.applet.Applet;

/*
<applet code = "Straitline.class" width= 400 height=300>
</applet>
*/

public class Straitline extends Applet
{
        public void paint(Graphics g)
        {
                g.drawLine(10,10,200,200);
        }
}
```

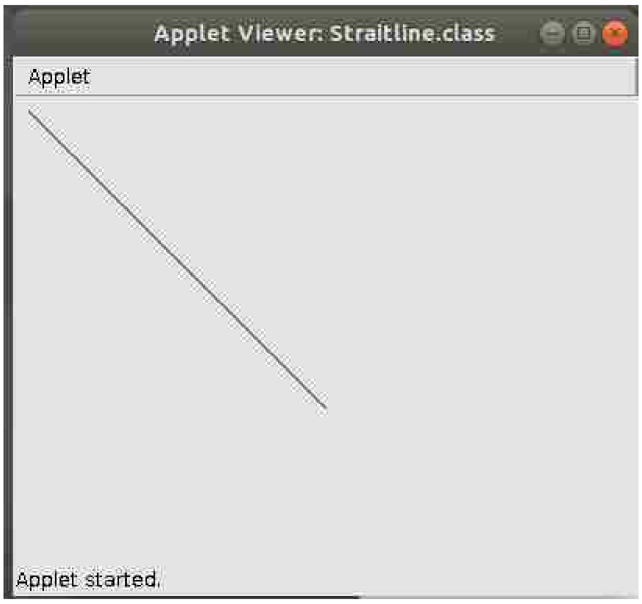


Figure 13.1 Output of Program

When we run the above program it will draw the line form (10, 10) coordinate to (200, 200) coordinates in applet.

13.3 Drawing Oval :

To draw the Oval in applet we need to use following method. The Oval also called as Ellipse.

Table 13.1 : Methods drawing the Oval or Circle

Method	Description
public void drawOval(int x, int y, int width, int height);	This method draws Ovals by rectangle with its top left corner(x,y) with the specified width and height. Here if the value of width and height are same then it will be draw circle.
public void fillOval(int x, int y, int width, int height);	This method draws Oval and fills with color. Here if the value of width and height parameters are same than it draws Circle.

Following program draws Empty Oval and filled Oval on applet :

```
import java.awt.Graphics;
import java.applet.Applet;

/*
<applet code = "Oval.class" width= 400 height=300>
</applet>
*/

public class Oval  extends Applet
{
        public void paint(Graphics g)
        {
                g.drawOval(10,25,100,130);
                g.fillOval(140,25,100,130);
        }
}
```

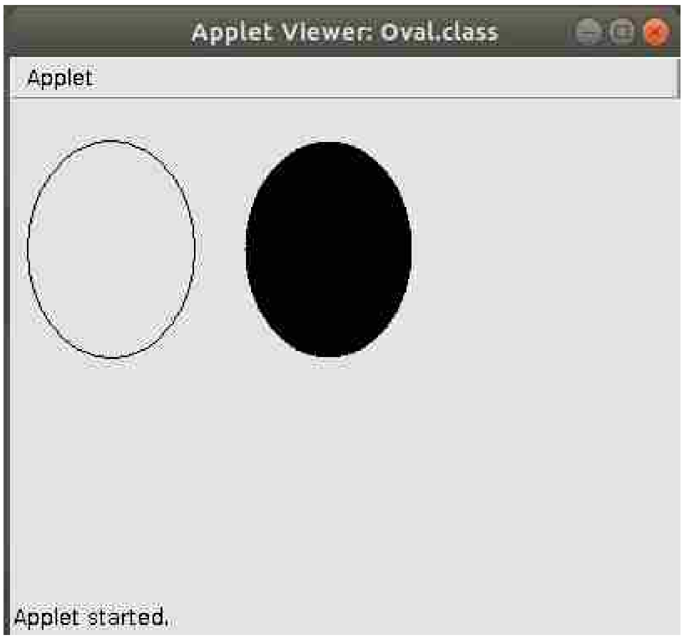


Figure 13.2 Output of Program

13.4 Drawing Circle :

Following program draws Empty Circle and filled Circle on applet. Please note that to draw the circle we need to use either 'drawOval( )' or 'fillOval( )' Methods. Here if the value of width and height are same in the said method then it will be draw circle.

```
import java.awt.Graphics;
import java.applet.Applet;

/*
<applet code = "Circle.class" width= 400 height=300>
</applet>
*/

public class Circle  extends Applet
{
      public void paint(Graphics g)
      {
            g.drawOval(10,25,100,100);
            g.fillOval(140,25,100,100);
      }
}
```

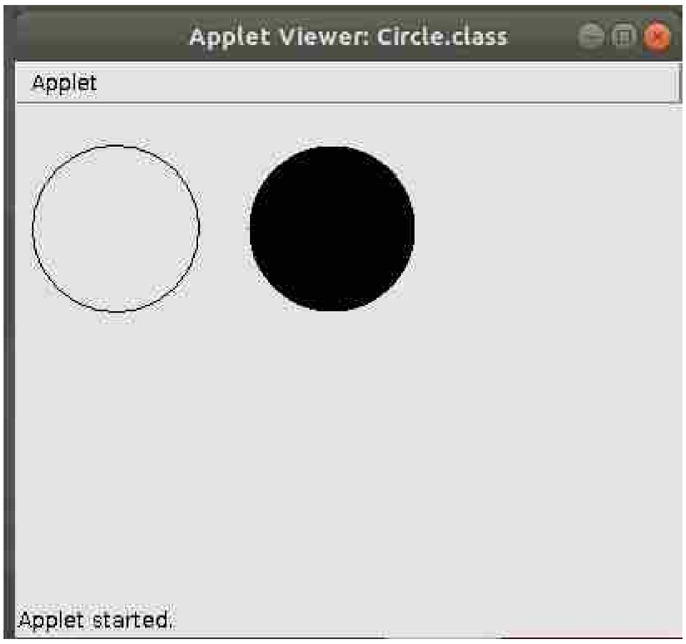


Figure 13.3 Output of Program

13.5 Drawing Rectangle :

Java provides various method to draw different kind of rectangles. To draw the Oval in applet we need to use following method.

Table 13.2 : Methods drawing the Rectangle or Square

Method	Description
public void drawRect(int x, int y, int width, int height);	This method draws rectangle with corner at (x,y) with specified width and height
public void drawRoundRect(int x, int y, int width, int height, int arcw, int arch);	This method draws rounded rectangle. First four parameters are same as above. And arcw is the horizontal diameter of the arc at the corners and arch is the vertical diameter of the arc.
public void draw3DRect(int x, int y, int width, int height, Boolean raised);	This method draws three dimensional rectangles. First four parameters are same as above and fifth parameter is raised. If raised is true, the rectangle appears to be raised above the surface.
pubic void fillRect(int x, int y, int width, int height);	This method draws rectangle and fills with color.
public void fillRoundRect(int x, int y, int width, int height, int arcw, int arch);	This method draws rounded rectangle and fill with color.
public void fill3DRect(int x, int y, int width, int height, Boolean raised);	This method draws three dimensional rectangles and fills with color.

Following program draws Empty Rectangle, filled Rectangle, Rounded Rectangle and 3D Rectangle on applet :

```
import java.awt.Graphics;
import java.applet.Applet;

/*
<applet code = "Rectangle.class" width= 400 height=400>
</applet>
*/

public class Rectangle extends Applet
{
        public void paint(Graphics g)
        {
                g.drawRect(10,25,100,130);
                g.fillRect(160,25,100,130);
                g.drawRoundRect(10,180,100,130,30,30);
                g.fill3DRect(160,180,100,130,false);
        }
}
```

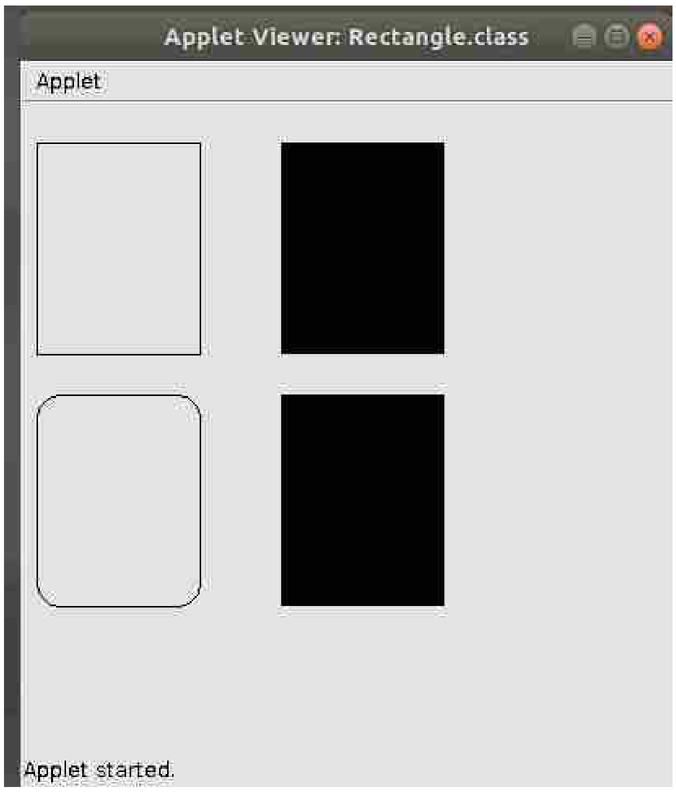


Figure 13.4 Output of Program

Following program draws Empty Square, filled Square, Rounded Square and 3d Square on applet. Please note that to draw the Square we need to use either 'darwRect( )' or 'fillRect( )' or 'darwRoundRect( )' or 'fill3DRect( )' Methods. Here if the value of width and height are same in the said method then it will be draw Square.

```
import java.awt.Graphics;
import java.applet.Applet;

/*
<applet code = "Square.class" width= 400 height=400>
</applet>
*/

public class Square extends Applet
{
        public void paint(Graphics g)
        {
                g.drawRect(10,25,100,100);
                g.fillRect(160,25,100,100);
                g.drawRoundRect(10,180,100,100,30,30);
                g.fill3DRect(160,180,100,100,false);
        }
}
```

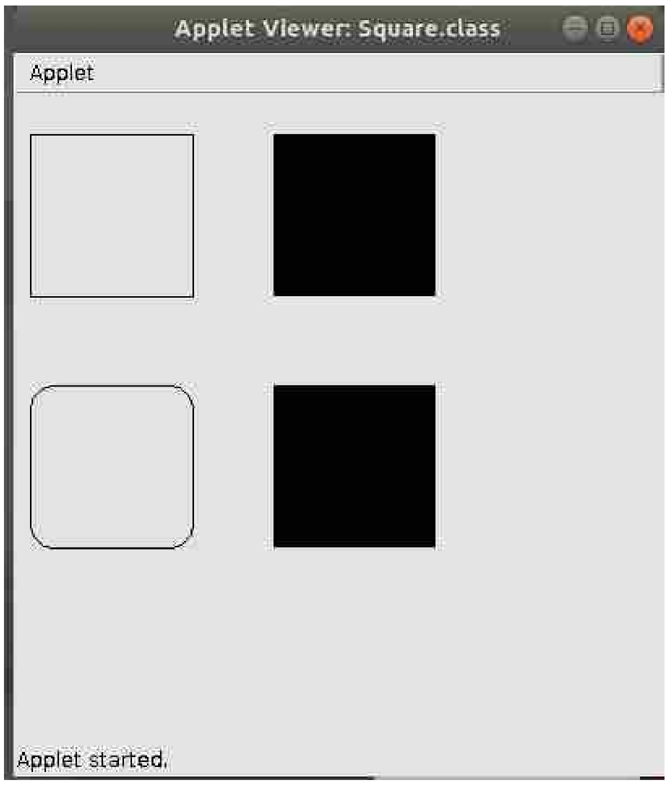


Figure 13.5 Output of Program

13.6 Drawing Arcs :

To draw the Arcs in applet we need to use following method. The co–ordinates to draw an arc are specified in bellow figure.

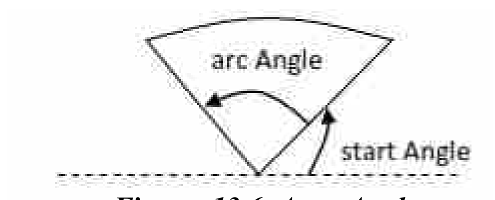


Figure 13.6 Arcs Angle

Table 13.3 : Methods drawing the Arcs

Method	Description
public void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle);	This method draws arc bounded by a rectangle having its top left corner at (x,y) with specified width and height. The arc starts with startAngle and sweeps an angle arcAngle.
public void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle);	This method draws arc and fills with color.

Following programs draws arc on applet.

```
import java.awt.Graphics;
import java.applet.Applet;

/*
<applet code = "Arc.class" width= 400 height=300>
</applet>
*/

public class Arc  extends Applet
{
        public void paint(Graphics g)
        {
                g.drawArc(10,25,100,130,10,140);
                g.fillArc(160,25,100,130,10,190);
        }
}
```

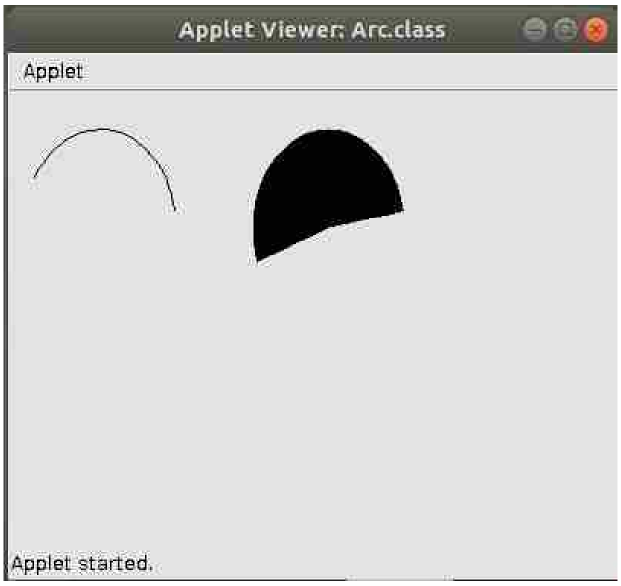


Figure 13.7 Output of Program

```
import java.awt.Graphics;
import java.applet.Applet;

/*
<applet code = "Arc1.class" width= 400 height=300>
</applet>
*/

public class Arc1  extends Applet
{
        public void paint(Graphics g)
        {
                g.drawArc(10,180,100,130,20,260);
                g.fillArc(160,180,100,130,20,260);
        }
}
```

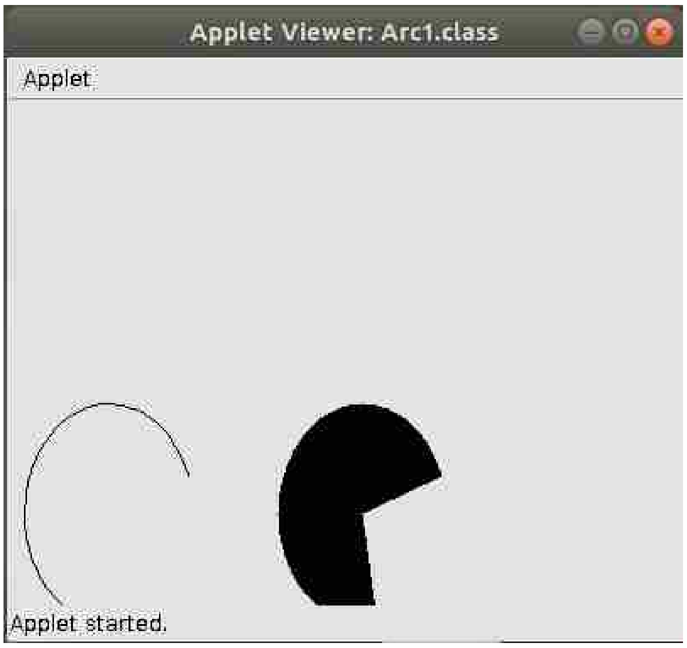


Figure 13.8 Output of Program

❑ **Check Your Progress – 1 :**

1. Explain various method which used to draw the rectangle.

....

....

....

....

....

13.7 Drawing Polygons :

Polygon is shape with many corners. Following are the some of the shape of Polygons.

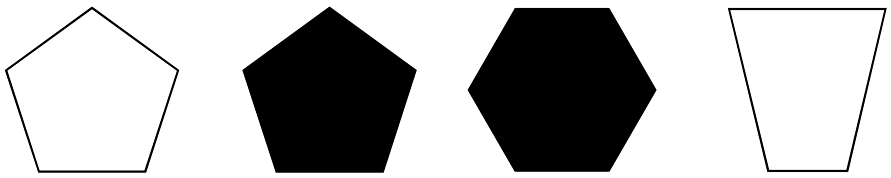


Figure 13.9 Different Shapes of Polygons

Polygons can be drawn using the following methods. Each method has two 'int' type arrays. One array is used for all the X co–ordinates and another array is used for the Y co–ordinates. Using X and Y co–ordinates polygon is forms. The close polygon is drawn using polygon methods. Any closed ended shape can be drawn using polygon method.

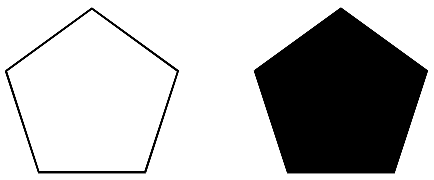
Table 13.4 : Methods drawing the Polygon

Method	Description
public void drawPolygon(int xPoints[], int yPoints[], int nPoints);	This method draws many corners of polygons. (x,y) point defined by xPoints[] and yPoints[]. nPoints represents the number of (x,y) pairs.
public void fillPolygon(int xPoints[], int yPoints[], int nPoints);	This method draws polygons and fills with color.

Following programs draws polygon on applet.

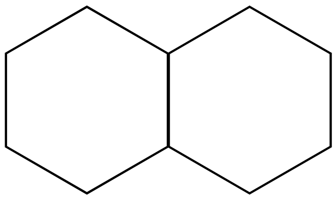
```
import java.awt.Graphics;
import java.applet.Applet;

/*
<applet code = "Polygon.class" width= 400 height=300>
</applet>
*/

public class Polygon  extends Applet
{
        public void paint(Graphics g)
        {
                int x[] = {50,10,30,70,90};
                int y[] = {10,40,90,90,40};
                g.drawPolygon(x,y,5);

                int x1[] = {150,110,130,180,200};
                int y1[] = {10,40,90,90,40};
                g.fillPolygon(x1, y1,5);
        }
}
```

The output of the above program is as follow :



```
import java.awt.Graphics;
import java.applet.Applet;

/*
<applet code = "Polygon1.class" width= 300 height=350>
</applet>
*/

public class Polygon1  extends Applet
{
    public void paint(Graphics g)
    {
        int x[] = {110,60,10,10,60,110,160,210,210,160,110,110};
        int y[] = {60,10,60,160,210,160,210,160,60,10,10,160};

        g.drawPolygon(x,y,12);
    }
}
```

The output of the above program is as follow :



13.8 Drawing Polyline :

The drawPolygon( ) method is draw the close ended polygon. If we want to draw open ended polygon then we need to use series of connected lines. The drawPolyline( ) method is used for drawing open ended polygon.

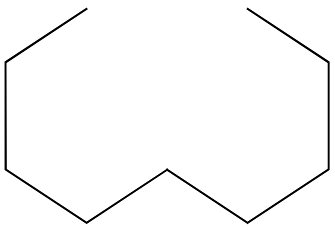
Table 13.5 : Methods drawing the Polyline

Method	Description
public drawPolyline(int xPoints[], int yPoints[], int nPoints);	Draws a sequence of connected lines, specified by the arrays xPoints, and yPoints, each pair(x,y) gives one point. The number of points for the polyline is nPoints.

Following programs draws polyline on applet.

```
import java.awt.Graphics;
import java.applet.Applet;

/*
<applet code = "Polyline.class" width= 300 height=350>
</applet>
*/

public class Polyline  extends Applet
{
      public void paint(Graphics g)
      {
            int x[] = {60,10,10,60,110,160,210,210,160};
            int y[] = {10,60,160,210,160,210,160,60,10};

            g.drawPolyline(x,y,9);
      }
}
```

The output of the above program is as follow :



❑ **Check Your Progress – 2 :**

1. Explain various method which used to draw the polygon.

   ....

   ....

   ....

   ....

   ....

2. ________ method is used to draw line.

   (A) drawLine( ) (B) Line( ) (C) StaritLine( ) (D) PlainLine( )

3. ________ method is used to draw Circle.

   (A) Oval( ) (B) Circle( )

   (C) drawCircle( ) (D) drawOval( )

4. To draw circle the value of heith and with should be same.

   (A) Fale (B) True

5. To draw rounded rectangle ________ method is used.

   (A) drawRect( ) (B) drawRoundRect( )

   (C) RoundRect( ) (D) Rect( )

6. To draw Rectangle the value of heith and with should be same. .

   (A) Fale (B) True

7. To draw Three dimension rectangle ________ method is used.

   (A) draw3DRect( ) (B) drawRect

   (C) 3DRect( ) (D) 3D( )

8. To draw fill Arc ________ method is used..

   (A) fillArc( ) (B) Arc( ) (C) drawArc( ) (D) solidArc( )

9. Polygon is shape with many corners.

   (A) Fale (B) True

10. The drawPolygon( ) method is draw the close ended polygon.

    (A) Fale (B) True

11. The drawPolygon( ) method is draw the open ended polygon .

    (A) Fale (B) True

13.9 Delegation Event Model :

In java, graphical user interface environment, action are initiated by the event. The Pressing a button, click of button, key press etc. are event. We required proper mechanisms that capture such events and to react to the events by executing a piece of code. Java provide such mechanisms called Delegation Event Model.

Delegation Event model is based on the concept of an 'Event source' and 'Event Listeners'. Any object that is interested in receiving event is called an event listener. Any object that generates these event is called an event sources.

Delegation event model is based on four concepts

1. The Event classes
2. The event listeners
3. Explicit event enabling
4. Adapters

Event in java are handled by delegation Event Model. In this model, there is a source, which generates events. There is a listener to the happenings of an event and initiate an action.

A listener has to register with a source. Any number of listeners can register with a source except in a few cases. A listener can register with many event sources when an event takes place. It is notified to the listener, which are registered with the source. The listener than initiates action.

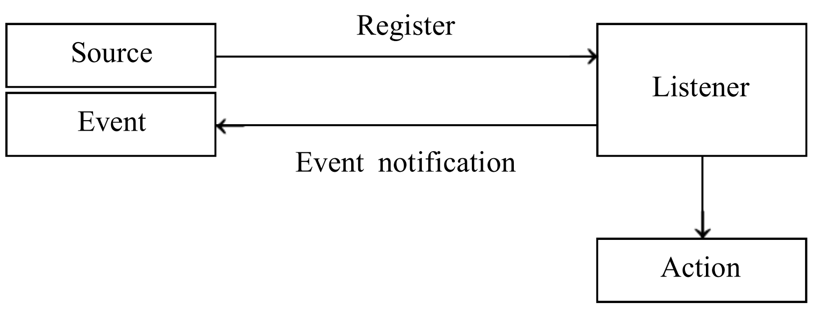


Figure 13.10 Delegation Event Model

Let discuss the event in detail. Now we know that an event is an object that inform the change of state of a source. Here we learn hierarchy of the event class. The super class of all events is 'java.util.EventObject'. Here we need to note that the super class of all AWT events is 'java.util.AWTEvent'.

AWTEvent class is an abstract class and contain subclasses. These subclasses are concrete and available under 'java.awt.event'.

Some of the classed mention in 'java.awt.event' package are describe in bellow table.

Table 13.5 : Event class defined under java.awt.event

Method	Description
Action Event	This event occurs when the component specific action takes place.
Adjustment Event	This event deals with events generated by the adjustable objects like scrollbar
Component Event	This event deals occurs when the component is moved, resized, visible or hide.
Item Event	This event generated when a check box or list iteam is clicked.
Key Event	This event generated when any key strokes.
Mouse Event	This event generated by Mouse
Text Event	This event generated by the change of object's text
Window Event	This event generated by the change of window status

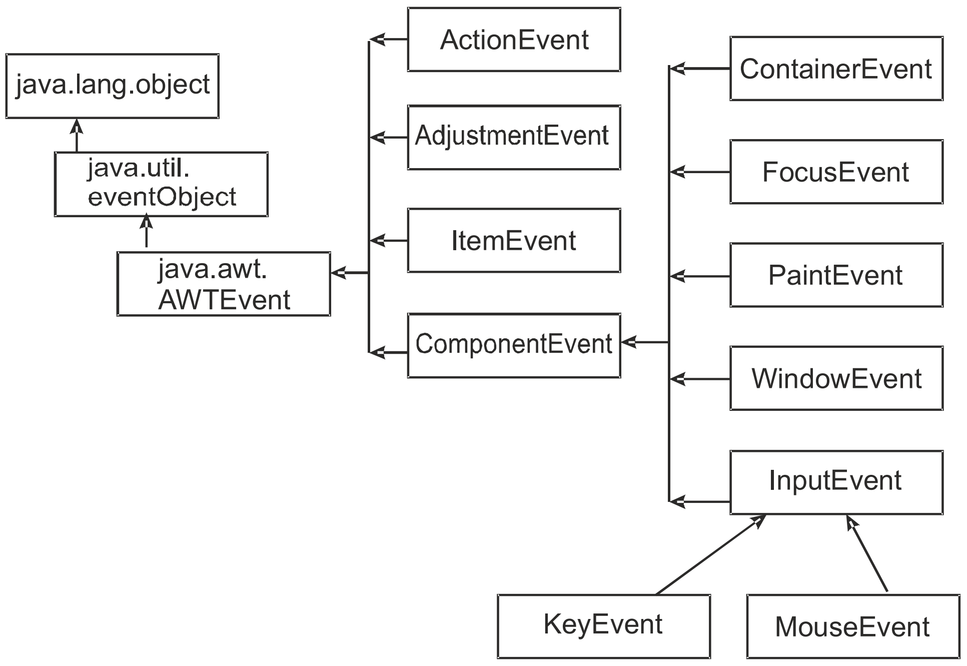


Figure 13.11 AWT Events

This section helps you learn handling events in Java AWT. Events are referred to as very intrinsic segments of the Java platform. The example below gives you a view of the concepts related to event handling and the methods which can be used to implement an application driven by events.

The objects register themselves as listeners for any event to occur. No event takes place if there is no listener, that is, nothing happens when an event takes place if there is no listener. Every listener has the capability of processing an event irrespective of how many listeners there are. For example, a Simple Button Event applet registers itself as the listener for the button's action events that creates a Button instance.

Any class including an applet can implement an Action Listener. You must always keep in mind that all the listeners are always notified. Moreover, in case you don't want any further processing of an event you can call AWT Event. consume () method. There is another method which is used by a listener to check for the consumption. The method is Consumed () method.

With the consumption of the events by the system once the listener is notified, the events stop being processed. Consumption only works for Input Event and its subclasses. You can use the consume () methods for the Key Event just in case you do not require any input by the user through the use of a keyboard.

The step by step procedure of Event handling is as follows :

1. The component generates subclasses of an AWT Event when anything interesting or intriguing takes place.
2. As permitted by the Event sources, any class can act like a Listener. For example, add ActionListener() method is used for any action to be performed, where Action is the event type. There is another method by which you can remove the listener class which is remove XXX Listener() method, where XXX is the event type.
3. A listener type such as an Action Listener must be implemented for handling an event.

4. Some special listener types require the implementation of multiple methods such as key Events. The key release, key typed and key press, are the three methods which are the requisites of implementation and registration of Key events. A few special classes exist referred to as adapters which are used in implementing listener interfaces and in stubbing out all the methods. Such adapter classes can be subclassed and necessary methods can be overridden.

AWT Event :

Most of the times every event–type has Listener interface as Events subclass the AWT Event class. However, Paint Event and Input Event do not have the Listener interface because only the paint() method can be overriden with Paint Event etc.

Low–level Events :

Low level events represent a low level input or window. Key press, mouse movement, window opening, etc. are the types of low level events.

For example, on typing the letter 'A', the three events one for pressing, one for releasing and one for typing are generated. Given below are the various types of low–level events and operations generated by each event.

FocusEvent	Used for Getting/losing focus.
MouseEvent	Used for entering, exiting, clicking, dragging, moving, pressing, or releasing.
ContainerEvent	Used for Adding/removing component.
KeyEvent	Used for releasing, pressing, or typing (both) a key.
WindowEvent	Used for opening, deactivating, closing, Iconifying, deiconifying, really closed.
ComponentEvent	Used for moving, resizing, hiding, showing.

Semantic Events :

The interaction with GUI component is represented by the Semantic events like changing the text of a text field, selecting a button etc. The different events generated by different components is shown below:

Table 13.6 : Events generated by different components

ItemEvent	Used for state changed.
ActionEvent	Used for do the command.
TextEvent	Used for text changed.
AdjustmentEvent	Used for value adjusted.

Event Sources :

If a component is an event source for something then the same happens with its subclasses. The different event sources are represented by the following table.

Table 13.7 : Event sources

Low–Level Events	
Window	WindowListener
Container	ContainerListener
Component	ComponentListener FocusListener KeyListener MouseListener MouseMotionListener
Semantic Events	
Scrollbar	AdjustmentListener
TextArea TextField	TextListener
Button List MenuItem TextField	ActionListener
Choice Checkbox Checkbox CheckboxMenuItem List	ItemListener

Event Listeners :

Every listener interface has at least one event type. Moreover, it also contains a method for each type of event the event class incorporates. For example as discussed earlier, the Key Listener has three methods, one for each type of event that the Key Event has: keyTyped(), keyPressed() and keyReleased().

The listener interfaces and their methods are as follow :

Table 13.8 : Listener interfaces and their methods

Interface	Methods
WindowListener	WindowActivated(WindowEvente)
	WindowDeiconified(WindowEvente)
	WindowOpened(WindowEvente)
	WindowClosed(WindowEvente)
	WindowClosing(WindowEvente)
	WindowIconified(WindowEvente)
	WindowDeactivated(WindowEvente)
ActionListener	ActionPerformed(ActionEvente)

AdjustmentListener	AdjustmentValueChanged (AdjustmentEvente)
MouseListener	MouseClicked(MouseEvente)
	MouseEntered(MouseEvente)
	MouseExited(MouseEvente)
	MousePressed(MouseEvente)
	MouseReleased(MouseEvente)
FocusListener	FocusGained(FocusEvente)
	FocusLost(FocusEvente)
ItemListener	ItemStateChanged(ItemEvente)
KeyListener	KeyReleased(KeyEvente)
	KeyTyped(KeyEvente)
	KeyPressed(KeyEvente)
ComponentListener	ComponentHidden(ComponentEvente)
	ComponentMoved(ComponentEvente)
	ComponentShown(ComponentEvente)
	ComponentResized(ComponentEvente)
MouseMotionListener	MouseMoved(MouseEvente)
	MouseDragged(MouseEvente)
TextListener	TextValueChanged(TextEvente)
ContainerListener	ComponentAdded(ContainerEvente)
	ComponentRemoved(ContainerEvente)

13.10 Let Us Sum Up :

In this Unit we have learned about applet graphics. To work with applet graphics we need to import 'java.awt.Graphics' package. To draw any shape we need required applet. The java applet can be import 'java.appelt.Applet'. To draw the strait line on applet we need to use drawLine( ) method. Applet graphics provide method for drawing Oval. To draw Oval, we need to used drawOval( ) method. Here in this method if the value of width and height are the same then this method draw perfect circle. If we want to draw coloured Oval then graphics provide fillOval( ) method. To draw rectangle we need to called drawRect( ) method. Applet graphics provides various method for rounded rectangle, fill rectangle and 3d rectangle. Here we also seen that if the height and width parameter's value of the rectangle are same then it became the square. Java graphics provide drawArc( ) method to draw the Arc of any shape. drawPolygon( ) method is used to draw various kind of polygon and drawPolyline( ) method is used to draw open ended polygon.

13.11 Suggested Answer for Check Your Progress :

❑ **Check Your Progress 1 :**

See Section 13.5

❑ **Check Your Progress 2 :**

1 : See Section 13.7		**2 :** A	**3 :** D
4 : B	**5 :** B	**6 :** A	**7 :** A
8 : A	**9 :** A	**10 :** A	**11 :** A

13.12 Glossary :

1. **Rectangle shape** – To draw the shape we can used drawRect(), Fill Rect( ), drawRoundRect( ) or draw3DRect( ) methods.
2. **Line** – To draw the shape we can used drawLine( )
3. **Oval or Circle** – To draw the shape we can used drawOval() or fillOval() methods
4. **Arcs** – To draw the shape we can used drawArc() or fillArc() methods.
5. **Polygons** – To draw the shape we can used drawPolygon() or fillPolygon() methods.
6. **PolyLine** – To draw the shape we can used drawPolyline() method.

13.13 Assignment :

1. Write a note Delegation Event Model.
2. Write a note that describe various method for the draw rectangle.
3. Explain to draw the polygons.

13.14 Activities :

1. Write a program to draw the two strait line.
2. Write a program that draw the square.
3. Write a program that draw the two Circle, one within other one.

13.15 Case Study :

1. Prepare the Chart that display various method to draw the different shapes.

13.16 Further Reading :

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000.
2. Java 2, the Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999.
3. Programming with Java, Ed. 2, E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000.
4. The Java Tutorial, Ed. 2, 2 volumes, Mary Campione and Kathy Walrath, Addison Wesley Longmans, 1998.
5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele & Gilad Bracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000.
6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition, 2000

Unit 14

ABSTRACT WINDOW TOOLKIT

UNIT STRUCTURE

14.0 Learning Objectives :

After learning this unit, you will be able to :

- Define window fundamentals.
- Explain working with graphics and controls.
- State layout managers and event handling.
- Describe adapter classes, inner classes and anonymous inner classes.
- Discuss applet fundamentals and applet lifecycle.

14.1 Introduction :

The Abstract Window Toolkit (AWT) is Java's original platform–independent windowing, graphics and user–interface widget toolkit. The AWT is now part of the Java Foundation Classes (JFC) the standard API for providing a graphical user interface (GUI) for a Java program.

AWT is also the GUI toolkit for a number of Java ME profiles. For example, Connected Device Configuration profiles require Java runtimes on mobile telephones to support AWT.

14.2 Window Fundamentals :

A top level window on the screen with no borders or menu bar is the provision of a window. Among various other provisions, it gives a way to implement pop–up messages. The default layout for a Window is Border Layout.

❖ Working with Frames :

A window which contains all the enhancements of the window manager such as borders, window title and window minimize/maximize/close functionality is called a frame. A frame can include a menu bar as well. The default layout of a frame is the Border Layout since it subclasses a window. The very basic building block of a screen–oriented application is provided by frames allowing you to change the mouse cursor, have menus and set icon images. The figure below is an example of a Frame.

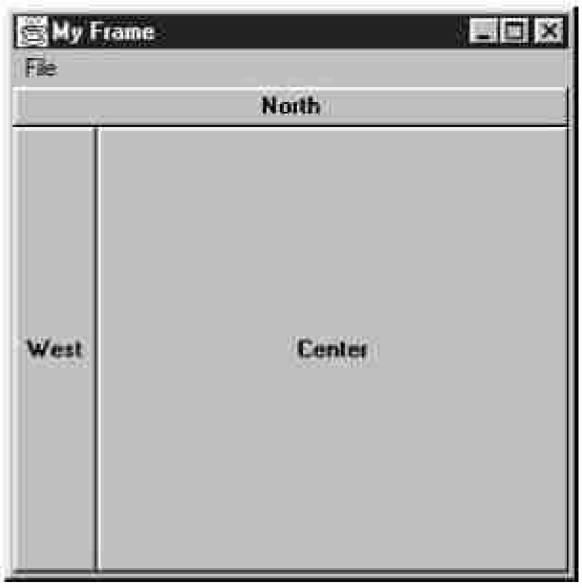


Figure 14.1 A Frame

A frame can be created with the help of a program in the Java AWT package. A frame in Java is similar to the main window into which components can be added or joined in order to develop an application. The Frame class represents the top–level windows in Java AWT. The feel and adornments of a frame are supported by Java. You must provide standalone for creating java application at the same time GUI to the user.

Generally, in order to create a frame the most familiar method used is the single string argument constructor of the frame class that contains a single string argument with just the title of the window frame. Later, a user interface can be added by constructing and joining various components to the container.

The program illustrated below shall construct a label on the message frame saying "–you are Welcome to the world of Java. – "The Label. CENTER defines the center alignment of the label. After creating the frame it needs to be visualized by set Visible (true) method as it is initially invisible.

add(lbl) :

This method has been used to add the label to the frame. Method add() adds a component to it's container.

setSize (width, height) :

This is the method of the Frame class that sets the size of the frame or window. This method takes two arguments width (int), height (int).

setVisible(boolean) :

This method of the Frame class sets the visibility of the frame. The frame will be invisible if you pass the boolean value false otherwise the frame will be visible.

Here is the code of the progam :

```
import java.awt.*
public class AwtFrame{
public static void main(String[] args){
Frame frm = new Frame("Java AWT Frame")
Label lbl = new Label("–you are Welcome to the world of Java.–",Label.CENTER)
frm.add(lbl)
frm.setSize(400,400)
frm.setVisible(true)
}
}
```

❑ **Check Your Progress – 1 :**

1. What is a frame ?
2. Write the use of addlbl().

....

....

....

....

....

14.3 Working with Graphics :

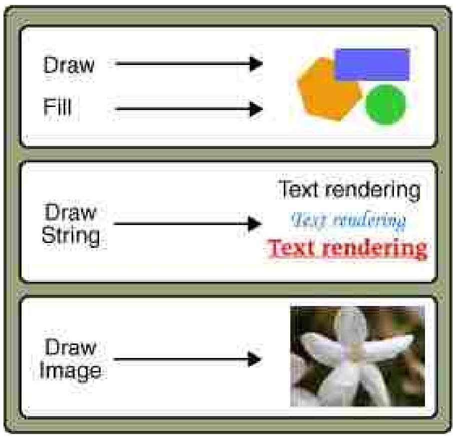


Figure 14.2 Applet Graphics

Java provides numerous primitives for drawing lines, squares, circles, polygons and images. The Font, Font Metrics, Color and System Color classes provide the ability to alter the displayed output. With the Font class, you adjust how the displayed text will appear. With Font Metrics, you can find out how large the output will be for the specific system the user is using. You could use the Color class to set the color of the text and graphics.

Whereas AWT also includes a number of classes that support more complex graphics manipulations: displaying images, generating images in memory and transforming images.

The Graphics class provides the means to access various graphics devises and is an abstract class. This class enables you to display images and draw on the screen. Since working with graphics requires a comprehensive knowledge of the platform that the program is run on it makes graphics an abstract class. Concrete classes that are closely tied to a particular platform are the ones which do the actual work. These concrete classes are provided by your Java Virtual Machine vendor.

You can be confident that the platform specific classes are going to work accurately whenever you run your program, since you can call all the methods of the Graphics class once you have a graphics object, leaving the worry about platform specific classes.

You rarely need to create a Graphics object yourself; its constructor is protected and is only called by the subclasses that extend Graphics. How then do you get a Graphics object to work with? The sole parameter of the Component.paint () and Component.update() methods is the current graphics context. Therefore, a Graphics object is always available when you override a component's paint () and update() methods.

You can ask for the graphics context of a Component by calling Component.getGraphics(). However, many components do not have a drawable graphics context. Canvas and Container objects return a valid Graphics object; whether or not any other component has a drawable graphics context depends on the run–time environment.

❑ **Check Your Progress – 2 :**

1. Why graphics is considered an abstract class ?
2. Give the use of canvas and container objects.

....

....

....

....

....

14.4 Controls :

Figure 14.3 Controls

Under this section you shall be informed about some components of Java AWT which are available in the Java AWT package for developing the user interface for your program.

1. **Labels –** the simplest component of the Java Abstract Window Toolkit is a label. Label does not perform any type of actions and is basically used to show a text or string in your application. Syntax for defining the label only and with justification:

   Label label_name = new Label ("This is the label text for the day")

   Above code simply represents the text for the label.

   Label label_name = new Label ("This is the label text for the day"

   Label.CENTER);

   The Justification of label can be right or left, centered. The Above declaration used the center justification of the label using the Label. CENTER.

2. **Buttons –** are utilized in generating actions and various other events required for an application. These are components of the Java Abstract Window Toolkit. The syntax of defining the button is as follows:

   Button button_name = new Button ("See this is the label of the button.")

   The Button.setLabel (String) and Button.getLabel() method can be used to change the Button's label or text. Buttons are added to its container using the add (button_name) method.

3. **Check Boxes –** allow you to create check boxes in applications. The syntax of the definition of Checkbox is as follows:

   CheckBox checkbox_name = new Checkbox ("Optional check box 1", false)

   The above code constructs the unchecked Checkbox by passing the boolean valued argument false with the Checkbox label through the Check box() constructor.

   Defined Checkbox is added to it's container using add (checkbox_name) method. You can change and get the checkbox's label using the setLabel (String) and getLabel() method. You can also set and get the state of the checkbox using the setState(boolean) and getState() method provided by the Checkbox class.

4. **Radio Button –** proves to be a special case of the Java AWT packages under the checkbox component where just a single checkbox from a group of checkbox can be selected at a time.

   Syntax for creating radio buttons is as follows :

   CheckboxGroup chkgp = new CheckboxGroup()

   add (new Checkbox ("1 One", chkgp, false)

   add (new Checkbox ("2 Two", chkgp, false)

   add (new Checkbox ("3 Three",chkgp, false)

   In the above code we are making three check boxes with the label "1 One", "2 Two" and "3 Three". If you mention more than one true valued for checkboxes then your program takes the last true and shows the last check box as checked.

5. **Text Area –** is the text container component of the Java AWT package. The Text Area contains plain text. TextArea can be declared as follows:

   TextArea txtArea_name = new TextArea()

   Now you can make the Text Area very much editable or not using the setEditable (boolean) method. If you pass the Boolean valued argument false then the text area will be non–editable otherwise it will be editable. The text area is by default in an editable mode. Text are set in the text area using the setText(string) method of the TextArea class.

6. **Text Field –** being a text container component of the Java AWT package, the text field component has single line and limited text information. This is declared as follows :

   TextField txtfield = new TextField(20)

   By specifying the number in the constructor you will be able to fix the number of columns in the text field. In the above code we have fixed the number of columns to 20.

❑ **Check Your Progress – 3 :**

1. Write a note on button.
2. Explain the text area component of the AWT package.

....

....

....

....

....

14.5 Understanding Layout Managers :

All Containers, by default, have a layout manager; an object that implements the Layout Manager interface. If a Container's default layout manager doesn't go well with your needs, you can easily replace it with another one. The AWT provide/ supplies layout managers that range from the very simple (FlowLayout and GridLayout) to the special purpose (BorderLayout and CardLayout) to the ultra–flexible (GridBagLayout).

General Rules for Using Layout Managers :

You have to openly tell a container not to use a layout manager, it is linked with its own occasion of a layout manager. This layout manager is automatically/by default consulted every time the Container might need to change its appearance. Most layout managers do not require programs to directly call their respective methods.

How to Choose a Layout Manager / Way :

Layout managers provided by AWT inherit various strengths and weaknesses. Under this section, common layout scenarios have been discussed and exactly which AWT layout managers might work for each situation and scenario has also been given. You can feel free to use layout managers contributed to the net, such as Packer Layout if you fail to feel that none of the AWT layout managers is suitable for your situation.

How to Create a Layout Manager and Associate It with a Container :

A default layout manager is associated with each container. Panels including Applets are initialized to use a Flow Layout whereas windows (apart from special purpose ones such as File Dialog) are initialized to use a Border Layout.

You don't have to worry about doing a thing to use a Container's default layout manager since each Container's constructor creates a layout manager instance and initializes the Container to use it.

You must create an instance (example) of the layout manager class that you require and tell the Container to use it in order to use a non default layout manager. A basic code that does this is mentioned below. This code creates a Card Layout manager and sets it up as the layout manager for a Container.

Container. set Layout (new Card Layout ())

Rules Of Thumb for Using Layout Managers :

The Container methods that result in calls to the Container's layout manager are add (), remove(), removeAll(), layout(), preferredSize() and minimumSize(). The add (), remove() and removeAll() methods add and remove Components from a Container; you can call them at any time. The layout() method, is called as the result of any paint request to a Container, requests that the Container place and size itself and the components it contains; you don't usually call it directly.

The preferredSize() and minimumSize() methods return the Container's ideal size and minimum size, respectively. Until your program does not enforce these sizes the values returned are merely hints.

You must take special care when calling a Container's preferred Size() and minimum Size() methods. Unless the container and its components have just peer objects the values returned by the methods are meaningless.

❖ **Border Layout :**

The default layout manager for all Windows is Border Layout, including Frames and Dialogs. North, south, east, west and center are the five areas it uses to hold components. All extra space is placed in the center area. Given below is an applet that places one button in each area.

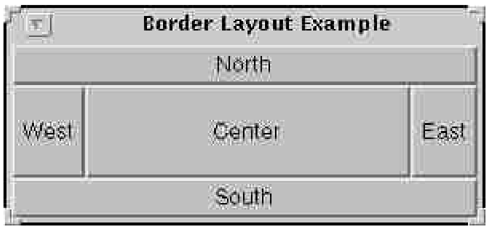


Figure 14.4 Border Layout

As the above applet shows, a Border Layout has five areas: north, south, east, west and center. You see that the center area gets the maximum of the new space that is available (likely) to it on enlarging the window. Whereas, with the other areas it is different since they expand only up to that space which is necessary to fill it up.

Below is the code that creates the Border Layout and the components it manages.

The program runs either within an applet, with the help of the Applet Button, or as an application. The first line shown below is actually unnecessary (not required) for this example, because it's in a Window subclass and each Window already has an associated Border Layout instance. However, the first line would be necessary if the code were in a Panel instead of a Window.

```
setLayout(new BorderLayout())
setFont(new Font("Helvetica", Font.PLAIN, 14))
add("–North–", new Button("–North–"))
add("–South–", new Button("–South–"))
add("–East–", new Button("–East–"))
add("–West–", new Button("–West–"))
add("–Cente–r", new Button("–Center–"))
```

By default, a Border Layout puts no space between the components it manages. In the applet above, any apparent gaps (space) are the result of the Buttons reserving extra space around their apparent display area. You can specify gaps (in pixels) using the following constructor:

```
Public BorderLayout(int horizontalGap, int verticalGap)
```

❖ **Card Layout :**

Let us Use the Card Layout class when you have an area that can contain different components at different times. Card Layouts are often controlled by Choices, with the state of the option (choice) determining which Panel (group of components) the Card Layout displays. Here's an applet that uses a Choice (option) and Card Layout in this way.

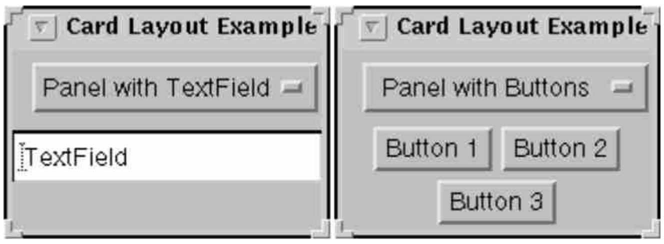


Figure 14.5 Card Layout

As illustrated in the applet above, the Card Layout class helps in managing two or more components (usually Panel instances) sharing the same display space. Each component managed by a Card Layout is hypothetically like a playing card or a trading card in a stack, with only the top card being visible at any one time. The card being displayed in any of the following manners can be chosen:

- By means of asking for either the first or last card, in the order they were added to the container.
- By means of flipping through the deck backwards or forwards.
- By means of specifying a card with a specific name.

The example program uses this scheme. The user can specifically choose (opt) a card (component) through selecting a name from a pop up list.

Below is the code that creates the Card Layout and the components it manages.

The program runs either within an applet, with the help of Applet Button, or as an application.)

```
//Where instance variables are declared :
Panel cards;
final static String BUTTONPANEL = "–Panel with Buttons–"
final static String TEXTPANEL = "–Panel with TextField–"

//Where the container is initialised
cards = new Panel();
cards.setLayout (new CardLayout())

...//–reate a Panel named p1. Put buttons in it.
.../–Create a Panel named p2. Put a text field in it.
cards.add(BUTTONPANEL, p1)
cards.add(TEXTPANEL, p2);
```

Adding a component to a container managed by a CardLayout uses the two–argument form of the Container add() method: add(String name, Component comp). Any string identifying the component that is added can be the first argument (case).

Below are all the Card Layout methods that let you choose(opt) a component. For each method, the first argument(case) is the container for which the Card Layout is the layout manager (the container of the cards the Card Layout controls).

```
public void first(–Container parent–)
public void next(–Container parent–)
public void previous(–Container parent–)
public void last(–Container parent–)
public void show(–Container parent, String name–)
```

❖ **Flow Layout :**

Flow Layout is the default (in any case) layout manager for all Panels. It simply lays out components from left to right, starting new rows if necessary.

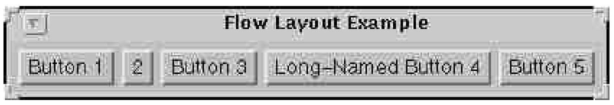


Figure 14.6 Flow Layout

Above shown the applet shows, Flow Layout puts components in a row, sized at their preferred size. If the horizontal space (gap) in the container is too small to put all the components in one row, Flow Layout uses multiple rows. Within each row, components are centered (the default), left–aligned, or right–aligned as specified when the Flow Layout is created.

You can see below is the code that creates the Flow Layout and the components it manages. The program runs either within an applet, with the help of Applet Button, or as an application.)

Set Layout (new FlowLayout ())

Set Font (new Font ("Helvetica", Font. PLAIN, 14))

Add (new Button ("Button –1–"))

Add (new Button ("2"))

Add (new Button ("Button –3–"))

Add (new Button ("Long–Named Button 4"))

Add (new Button ("Button 5"))

The Flow Layout class has three constructors:

public Flow Layout()

public Flow Layout(int alignment)

public Flow Layout(int alignment, int horizontal Gap, int vertical Gap)

The alignment argument must have the value Flow Layout. LEFT, Flow Layout. CENTER, or Flow Layout. RIGHT. The horizontal Gap and vertical Gap arguments specify the number of pixels to put between components. If you don't specify a gap value, Flow Layout acts as if you specified 5 for the gap value.

❖ **Grid Layout :**

Grid Layouts simply make a bunch of Components have equal size, displaying them in the requested number of rows and columns. Here's an applet that uses a Grid Layout to control the display of five buttons.

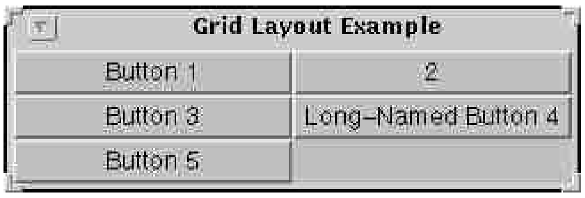


Figure 14.7 Grid Layout

Illustrated in the above applet is a Grid Layout placing its components in a grid of cells. Each component takes all the available space within its cell and each cell is exactly the same size. Given the space available to the container, you'll notice that the Grid Layout will change the size of cells so that they are as large as possible once you resize the Grid Layout window.

Below is the code that creates the Grid Layout and the components it manages.

The constructor tells the Grid Layout class to create an instance that has two columns and as many rows as necessary. It's one of two constructors for Grid Layout. Here are the declarations for both constructors:

public Grid Layout(int rows, int columns)

public Grid Layout(int rows, int columns,

int horizontalGap, int verticalGap)

Among all the arguments at least one must be non zero. The numbers of pixels among the cells are specified with the help of the horizontal Gap and vertical Gap arguments. The values shall default to zero if the gaps are not specified. (In the applet above, any apparent gaps are the result of the Buttons reserving extra space around their apparent display area.)

❖ Grid Bag Layout :

Grid Bag Layout is the most sophisticated, flexible layout manager the AWT provides. It aligns components by placing them within a grid of cells, allowing some components to span more than one cell. The rows in the grid are not necessarily all the same height; similarly, grid columns can have different widths. Here is an applet that uses a Grid Bag Layout to manage ten buttons in a panel.

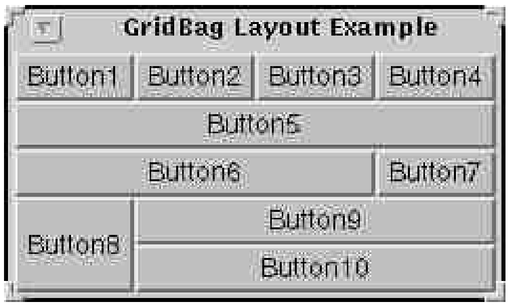


Figure 14.8 GridBag Layout

This layout is the most flexible yet complex layout manager that the AWT provides. The Grid Bag Layout, as illustrated in the applet above places the components in a grid of rows and columns, allowing certain components to span multiple rows or columns. It is not necessary for each row to have the same height just as each column does not have the same width. Essentially, Grid Bag Layout places components in squares (cells) in a grid and then uses the components' preferred sizes to determine how big the cells should be.

You notice that once you enlarge the window brought up by the applet, it results in the last row getting all the new available vertical space whereas the horizontal space gets evenly distributed among all the columns. The weight that the applet assigns to individual components in the Grid Bag Layout determines the resizing behavior. Notice that each component takes the maximum space it can and this is also specified by the applet.

By specifying the constraints for each component an applet specifies their size and characteristics. To specify constraints, you set instance variables in a Grid Bag Constraints object and tell the Grid Bag Layout (with the set Constraints() method) to associate the constraints with the component.

❑ Check Your Progress – 4 :

1. Give the general rules for using Layout Manager.
2. Explain Card Layout.

....

....

....

....

....

14.6 Adapter Classes :

There are some event listeners that have multiple methods to implement. That is some of the listener interfaces contain more than one method. For instance, the Mouse Listener interface contains five methods such as mouse Clicked, mouse Pressed, mouse Released etc. If you want to use only one method out of these then also you will have to implement all of them. Hence, the methods which we do not want to care about can have empty (clear) bodies. To avoid such thing, we have adapter class.

Now it is fact that Adapter classes help us in avoiding the implementation (execution of) of the empty method bodies. Usually (Generally) an adapter class is there for each listener interface having more than one method.

For example, the Mouse Adapter class implements the Mouse Listener interface. An adapter class can be used by creating (making) a subclass of the same and then overriding the methods which are of use only. Hence, it avoids the implementation of all the methods of the listener interface. The following example shows the implementation of a listener interface directly.

```
public class MyClass implements MouseListener {
...
someObject.addMouseListener(this);
...
/* –Empty method definition– ok. –*/
public void mouseEntered(MouseEvent e) {
}

/*– Empty method definition–ok –. */
public void mouseExited(MouseEvent e) {
}
/*– Empty method definition–ok– */
public void mousePressed(MouseEvent e) {
}


}
}
```

In the above program code, the adapter class has been used (untilled). This class has been used as an anonymous (unknown) inner class to draw a rectangle within an applet. This example demonstrates the functionality of the mouse press. That is on every click of the mouse from top left corner, we are going to get a rectangle on the release of the bottom right.

❑ **Check Your Progress – 5 :**

1. What are adapter classes ?
2. Give the use of adapter classes

....

....

....

....

....

14.7 Inner Classes :

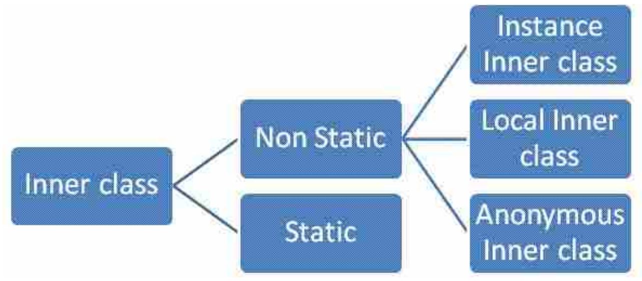


Figure 14.9 Inner Classes

Inner classes cannot have (will not have) static members, only static final variables. Interfaces are never be inner. Hence Static classes are not inner classes.

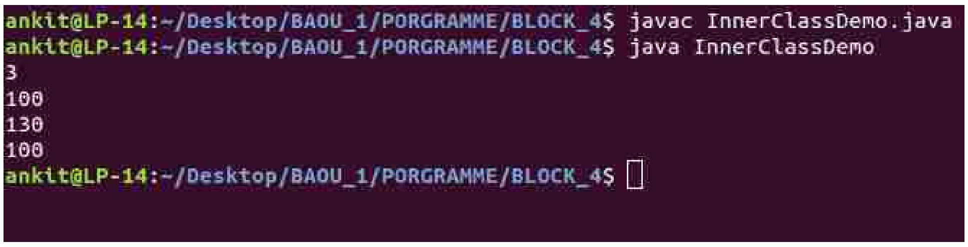
Inner classes may inherit static members that are not compile–time constants even though they may not declare them.

Nested classes that are not inner classes may declare static members freely, in accordance with the usual rules of the Java programming language. Member interfaces are always implicitly static so they are never considered to be inner classes. A statement (agreement) or expression occurs in a static context if and only if the innermost method, constructor, instance initialiser, static initialiser, field initialiser, or explicit constructor invocation (spell) statement enclosing the statement or expression is a static method, a static initialiser, the variable initialiser of a static variable or an explicit constructor invocation (spell) statement.

For Example:

```
class HasStatic
{
 static int j = 100;
}

class Outer
{
 final int z=10;
 class Inner extends HasStatic
 {
  static final int x = 3;
```

```
   static int y = 4;
  }

  static class Inner2
  {
   public static int size=130;
  }

  interface InnerInteface
  {
   public static int size=100;
  }
 }

 public class InnerClassDemo
 {
  public static void main(String[] args)
  {
   Outer outer=new Outer();
   System.out.println(outer.new Inner().z);
   System.out.println(outer.new Inner().x);
   System.out.println(outer.new Inner().j);
   System.out.println(Outer.Inner2.size);
   System.out.println(Outer.InnerInteface.size);
  }
 }
```



```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_4$ javac InnerClassDemo.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_4$ java InnerClassDemo
3
100
130
100
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_4$ 
```

Figure 14.10 Output of Program

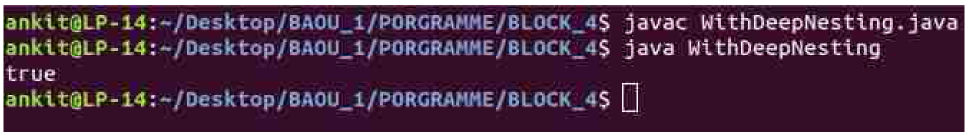
Hence, in this case it gives compilation problems, as z cannot be used in inner class "Inner".

Also note that a Method parameter names may not be redeclared as it is a local variables of the method, or you may say as exception parameters of catch clauses in a try statement of the method or constructor. However, a parameter of a method or constructor may be shadowed anyplace inside a class declaration nested within that method or constructor. Such a nested class declaration (statement) could declare either a local class or an anonymous class.

Now coming to Section Nested Inner Classes :

Consider the below program:

```
class WithDeepNesting
{
 boolean toBe;
 WithDeepNesting(boolean b) { toBe = b;}

 class Nested
 {
  boolean theQuestion;
  class DeeplyNested
  {
   DeeplyNested(){
    theQuestion = toBe || !toBe;
      }
  }
 }
 public static void main(String[] args)
 {
  WithDeepNesting withDeepNesting=new WithDeepNesting(true);
  WithDeepNesting.Nested nested=withDeepNesting.new Nested();
  nested.new DeeplyNested();
  System.out.println(nested.theQuestion);
 }

}
```



```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_4$ javac WithDeepNesting.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_4$ java WithDeepNesting
true
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_4$ 
```

Figure 14.11 Output of Program

We may also note and understand that the Inner classes whose declarations do not occur in a static context may freely refer to the instance variables of their enclosing (attached) class. We need to understand an instance variable is always defined (clear) with respect to an instance. Now in the case of instance variables of an enclosing class, the instance variable must be defined with respect to an enclosing instance of that class. So, we can consider the example, the class Local above has an enclosing instance of class Outer. As a further example we can say that:

In this every instance of With Deep Nesting. Nested. Deeply Nested has an enclosing instance of class With Deep Nesting. Nested (its immediately enclosing instance) and an enclosing instance of class With Deep Nesting (its 2nd lexically enclosing instance). Hence, it is going to prints : true.

❑ **Check Your Progress – 6 :**

1. Write a note on inner classes.
2. What could be declared a local or anonymous class ?

....

....

....

....

....

14.8 Anonymous Inner Classes :

An Inner classes have made it possible to code with less lines, coding became easy task and in the process to obscure the code to the unenlightened but make it wonderfully well–designed (elegant) to those who understand. A good friend of mine and coworker told me "we may say that One good Java programmer is better than comparatively about ten bad Java programmers" which I heartily agreed with. He then went on to say "And five bad Java programmers are better than ten bad Java programmers". Hence Inner classes have widened this divide. We can say that the typical way of using anonymous inner classes is for writing GUI event handlers, let us take for example,

```
button.addActionListener (new ActionListener () {
//      This is how we define an anonymous inner class
public void actionPerformed(ActionEvent e) {
System.out.println ("–ok the button was pressed!–")
}
});
```

The amazing thing is that we are actually defining a new class (!) while calling another method. You can virtually make new classes in all sorts of places in Java.

```
new Thread(new Runnable() {
public void run() {
try {
while (true) {
sleep(1000); System.out.print(".")
}
}
catch(InterruptedException ex) { }
}
}).start()
```

If we try to look at the definition of a Thread, we can see that it takes a Runnable (to work) as a parameter so it makes sense to create an anonymous inner class from Runnable (to work) and stick (fix in to) that into the parameter. However, after looking more carefully inside Thread we notice that the run () method defined in Thread, calls the run() method defined in Runnable, if a Runnable has been passed into the Thread constructor. Instead, it would be more efficient to do the following, because we would have one less object on the heap and one less method call per Thread creation:

```
new Thread() {
public void run() {
try {
while (true) {
sleep(1000); System.out.print(".")
}
}
catch(InterruptedException ex) { }
}
}).start()
```

This way Thread by itself is made into an anonymous inner class and we override the run() method so as an alternative (in place of) of Thread.run() having to check that a Runnable exists (present) we can just execute the code in run().

An application (use) of anonymous inner class is to pass an array as a parameter to a method, which is shown below statements :

```
String[] temp_names = new String[4]
temp_names[0] = "Ashok" temp_names[1] ="Zameer"
temp_names[2] = "Monica"
temp_names[3] = "Shakila"
universityRegistration.addNames(temp_names)
```

or, alternatively

```
String[] temp_names = {"Ashok","Zameer" , "Monica","Shakila" }
universityRegistration.addNames(temp_names)
```

Know we would not need to have a temporary variable which is bad. We could thus say that:

```
universityRegistration.addNames(
new String[]({"Ashok","Zameer" , "Monica","Shakila" })
```

If you wanted to pass in a Collection instead of an array it would look as follows :

```
Collection temp_names = new Vector(4)
temp_names.add("Ashok")
temp_names.add("Zameer")
temp_names.add("Monica")
temp_names.add("Shakila")
universityRegistration.addNames(temp_names);
```

Here the ability to avoid local temporary variables with an arrays was always (every time) a strong deciding (decision making) factor in defining interfaces to my classes because we could get away with(replace with) one line of code instead of five and the less lines of code the better hence, with anonymous inner classes we can get the same effect seen above but with collections:

```
universityRegistration.addNames (new Vector(4)
{{ add Ashok","Zameer" , "Monica","Shakila" }})
```

Hence the call to the super constructor always takes place first, so we could re–write MyVector as follows without changing the functionality in any way:

```
public class MyVector extends Vector {
{ // initialiser block
  add ("Ashok","Zameer" , "Monica","Shakila");
}
public MyVector() {
  super(4); // to initialise it with a size of 4
  }
}
```

In case if we want to make an instance of an anonymous inner class we can pass the parameters directly to the super class via the parameter list of the constructor of the anonymous class. In addition to this, any init block denoted by

{ } is done AFTER the call to the super class constructor is completed, so the class MyVector could look like this:

```
Vector myVector =
  new Vector(4) { // defining anonymous inner class
{
add;
}
};
```

From here the step to addNames(new Vector(4) {{ add("Ashok"); add("Zameer"); add("Monica") add("Shakila");}});

❑ **Check Your Progress – 7 :**

1. Define an anonymous inner class.
2. Explain how thread itself is made into an anonymous inner class.

....

....

....

....

....

3. A window which contains all the enhancements of the window manager is called a ________.

   (A) Window (B) Box (C) Contorl (D) Frame

4. After creating the frame it needs to be visualized by ________ method.

   (A) show() (B) setVisible()

   (C) setVisible(true) (D) Visible( )

5. The simplest component of the Java Abstract Window Toolkit is a ________.

   (A) TextField (B) Frame (C) Button (D) label

6. Multiselection can be possible using ________ control.

   (A) CheckBox (B) RadioBox (C) TextFiled (D) TextArea

7. Single selection using multiselection can be possible using ________ control .

   (A) CheckBox (B) RadioBox (C) TextFiled (D) TextArea

8. The ________ control contains plain text.

   (A) CheckBox (B) RadioBox (C) TextFiled (D) TextArea

9. The default layout manager for all Windows is ________.

   (A) Flow Layout (B) Box Layout

   (C) Grid Layout (D) Border Layout

10. ________ layout is used when you have an area that can contain different components at different times.

    (A) Card Layout (B) Box Layout

    (C) Grid Layout (D) Border Layout

11. Adapter classes help us in avoiding the implementation of the empty method bodies.

    (A) False (B) True

12. Inner classes cannot have static members.

    (A) False (B) True

14.9 Let Us Sum Up :

This Unit made us learn about Abstract Window Toolkit which is nothing but is Java's original platform–independent windowing, graphics and user–interface widget toolkit. The AWT is now part of the Java Foundation Classes (JFC) the standard API for providing a graphical user interface (GUI) for a Java program. A Window provides a top–level window on the screen, with no borders or menu bar. It provides a way to implement pop–up messages, among other things. The default layout for a Window is Border Layout.

Working with Frames : A Frame is a Window with all the window manager's adornments (window title, borders, window minimize/maximize/close functionality) added. It may also include a menu bar. Since Frame subclasses Window, its default layout is Border Layout. Frame provides the basic building block for screen–oriented applications.

As long as working with Graphics is concern Java provides numerous primitives for drawing lines, squares, circles, polygons and images. The figure shows a simple drawing. The Font, Font Metrics, Color and System Color classes provide the ability to alter the displayed output. With the Font class, you adjust how displayed text will appear. With Font Metrics, you can find out how large the output will be, for the specific system the user is using. You could use the Color class to set the color of text and graphics.

We have learned about Controls where in Labels, Buttons, Check Boxes, Radio Button, Text Area, and Text Field covered. After this we learned about Layout Manager. In this All Containers, by default, has a layout manager; an object that implements the Layout Manager interface. If a Container's default layout manager doesn't go well with your needs, you can easily replace it with another one. We further came to know about Rule those can be said as you have to openly tell a container not to use a layout manager, it is linked with its own occasion of a layout manager. This layout manager is automatically/ by default consulted every time the Container might need to change its appearance. Most layout managers do not require programs to directly call their respective methods.

There is also learning about how to choose A Loyout Manager /Way and D How ti Reate A reate a Layout Manager and Associate it with a container.

We also seen and understood about rules for Layout Manager like Flow layout, Grid Layout and Grid bag layout. Further we have covered event handling.

Events are the integral part of the java platform. You can see the concepts related to the event handling through the example and use methods through which you can implement the event driven application.

There was also learning related to the step by step procedure of Event handling and even Events and the operations that generate them and Events generated by different components.

Further we have also learned some event listeners that have multiple methods to implement. That is some of the listener interfaces contain more than one method. For instance, the Mouse Listener interface contains five methods such as mouse Clicked, mouse Pressed, mouse Released etc. There is also mention about Inner classes cannot have static members, only static final variables. 1. Interfaces are never inner. 2. Static classes are not inner classes. We have also learned about anonymous inner classes. There is in detail learning about applets, applets life cycle

14.10 Suggested Answer for Check Your Progress :

❑ **Check Your Progress 1 :**

See Section 14.2

❑ **Check Your Progress 2 :**

See Section 14.3

❑ **Check Your Progress 3 :**

See Section 14.4

❑ **Check Your Progress 4 :**

See Section 14.5

❑ **Check Your Progress 5 :**

See Section 14.6

❑ **Check Your Progress 6 :**

See Section 14.7

❑ **Check Your Progress 7 :**

1 : See Section 14.8		**2 :** See Section 14.8		**3 :** D
4 : C	**5 :** D	**6 :** A	**7 :** B	**8 :** D
9 : D	**10 :** A	**11 :** B	**12 :** B	

14.11 Glossary :

1. **Frame –** A Frame is a Window with all the window manager's adornments (window title, borders, window minimize/maximize/close functionality) added.
2. **Inner classes –** Inner classes may inherit static members that are not compile–time constants even though they may not declare them.
3. **Adapter class –** methods which you do not want to care about can have empty bodies. To avoid such thing, we have adapter class.

14.12 Assignment :

1. Write a program to create Grid Layout.
2. Write the rules of thumb for using Layout managers.

14.13 Activities :

Write programs to illustrate the use of controls?

14.14 Case Study :

A case study on Java Applet: You have to write a Java applet to calculate the average of a list of integers and their count. Now the input data is stored in a text file, to this each line contains a single integer. The input file is chosen via Open File Windows–like Dialog box. In this the average and count (number of integers in a file) are displayed inside the applet. Now you provide for error checking of the file name, i.e. the program must display a meaningful message when the input file name is incorrect, and it should ignore (non consideration) any lines in the input file that are other than a single integer.

14.15 Further Reading :

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000
2. Java 2, The Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999

3. Programming with Java, Ed. 2,E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000

4. The Java Tutorial, Ed. 2, 2 volumes, Mary Campione and Kathy Walrath, Addison Wesley Longmans, 1998

5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele & Gilad Bracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000

6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition, 2000

Unit 15

WORKING WITH FILES

UNIT STRUCTURE

15.0 Learning Objectives :

After learning this unit, you will be able to :

- Define i/o streams
- Describe reading console input and writing console output
- Explain reading and writing files
- Discuss serialization

15.1 Introduction :

The Java I/O is said as Java Input /Output and is a part of java.io package. Now package has an Input Stream and Output Stream. The Java Input Stream is meant for reading the stream, byte stream and array of byte stream. This can be used for memory allocation. The Output Stream is used for writing byte and array of bytes.

In this lesson, we will learn and understand streams that can handle all kinds of data, from primitive values to advanced objects.

15.2 I/O Streams :

An input source or an output target is represented (shown) by an I/O Stream. A stream can represent disk files, devices, other programs and memory arrays and also as many different kinds of sources and destinations target.

Streams support simple bytes, primitive data types, localized characters and objects as well as many different kinds of data. Some streams simply pass on data while the others manipulate and transform the data in useful ways.

No matter how they work internally, all streams present the same simple model to programs that use them: A stream is a sequence of data. A program uses an input stream to read data from a source, one item at a time :

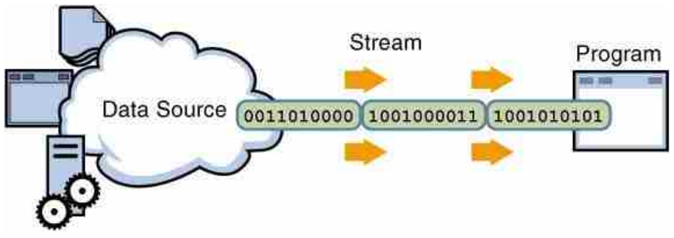


Figure 15.1 Reading Information into a Program

A program uses an output stream to write data to a destination, one item at time :

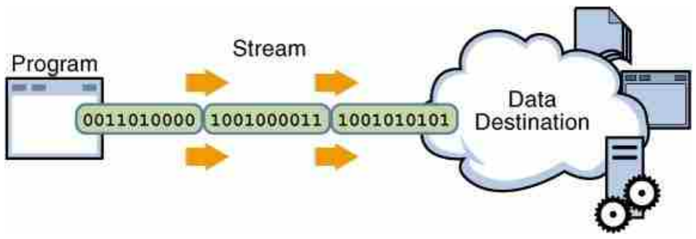


Figure 15.2 Writing Information from a Program

The data source and data destination pictured in the figure can be anything that stores, generates or consumes data. Obviously, this comprises of disk files but a source or destination can also be another program, a peripheral device, a network socket or an array.

❑ **Check Your Progress – 1 :**

1. Explain I/O stream.

....

....

....

....

....

15.3 Streams :

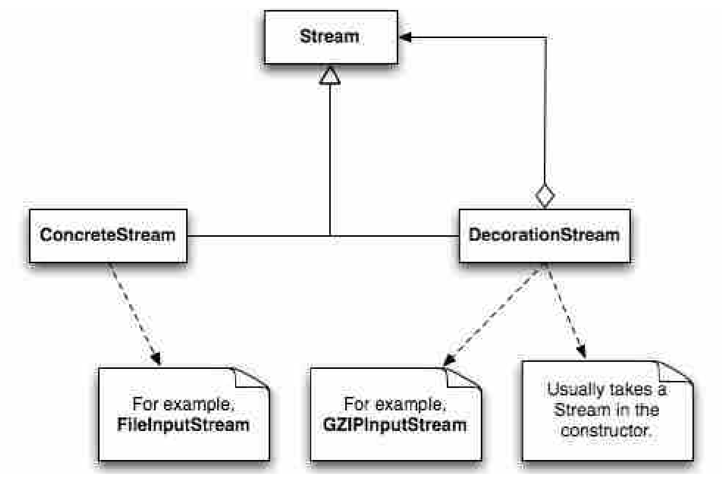

Stream
ConcreteStream
DecorationStream
For example, FileInputStream
For example, GZIPInputStream
Usually takes a Stream in the constructor.

Figure 15.3 Stream

❖ **Byte Streams :**

The Byte streams are used by programs for the purpose of performing input and output of 8–bit bytes. Here all byte stream classes are descended from Input Stream and Output Stream.

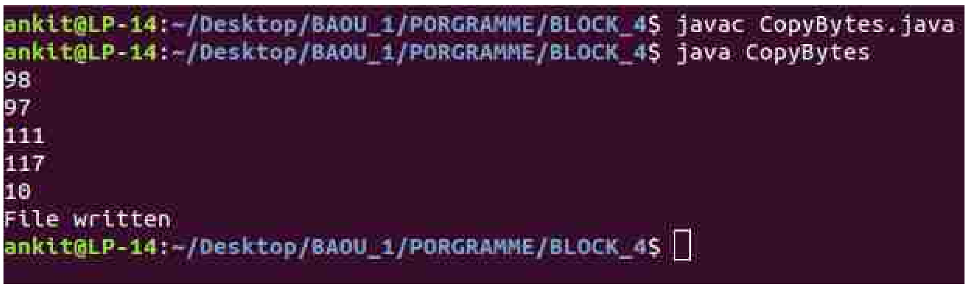
We must understand that there are many byte stream classes. To demonstrate how byte streams work, we'll focus on the file I/O byte streams, File Input Stream and File Output Stream. Now other kinds of byte streams are used in a lot the same way; they differ mainly in the way they are constructed or made.

❖ **Using Byte Streams :**

Now we will search File Input Stream and File Output Stream by examining an example program named Copy Bytes, which uses byte streams to copy hello.txt, one byte at a time.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class Copy Bytes
{

 public static void main (String[] args) throws IOException
 {
  FileInputStream in = null;
  FileOutputStream out = null;

  try
  {
   in = new FileInputStream("hello.txt");
   out = new FileOutputStream("outagain.txt");
   int c;

   while ((c = in.read()) != -1)
   {
    out.write(c);
   }

  }
  finally
  {
   if (in != null)
   {
    in close();
```

```
      }

      if (out != null)
      {
       out. close();
      }
     }

    }
   }
```



```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_4$ javac CopyBytes.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_4$ java CopyBytes
98
97
111
117
10
File written
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_4$ 
```

Figure 15.4 Output of Program

Copy Bytes spends most of its time in a simple loop that reads the input stream and writes the output stream, one byte at a time, as shown in the following figure.

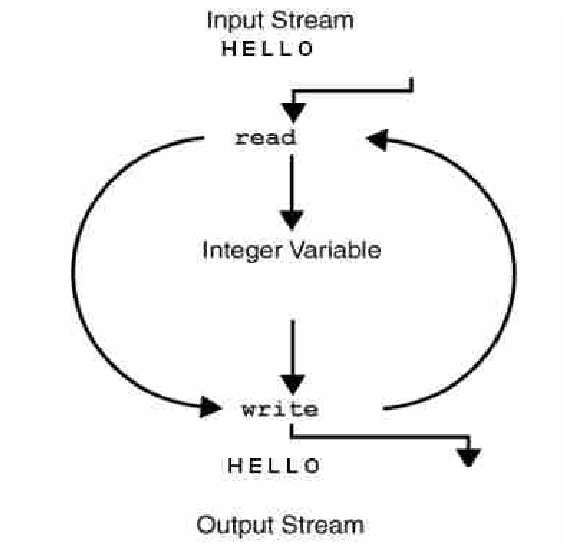


Figure 15.5 Simple Byte Stream Input and Output

Notice that read () returns an int value. If the input is a stream of bytes, why does not read () return a byte value? Using an int as a return type allows read () to use –1 to indicate that it has reached the end of the stream.

❖ **Always Close Streams :**

It is very important to close a stream when it is no longer needed. Copy Bytes uses a finally block to guarantee that both streams will be closed even if an error occurs. This practice helps to avoid serious resource leaks.

Copy Bytes being unable to open one or both files is one of the possible errors found. When that happens, the stream variable corresponding to the file never changes from its initial null value. For this reason, Copy Bytes makes sure that each stream variable contains an object reference before invoking close.

❖ **Alternate of Byte Streams :**

Copy Bytes seems like a normal program but it actually a representation of a kind of low–level I/O that you should avoid. Since hello.txt contains character data, the best approach is to use character streams, as discussed in the next section. There are also streams available for more complicated data types. Byte streams should only be used for the most primitive I/O.

So why talk about byte streams? As all other stream types are built on byte streams.

❖ **Character Streams :**

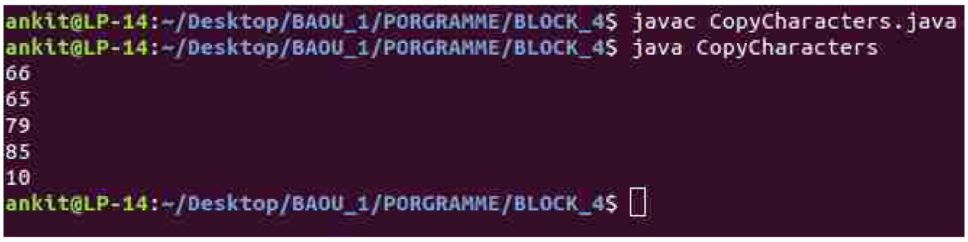
The character values are stored by Java platformusing Unicode conventions. Character stream I/O automatically translates this internal format to and from the local character set. In Western locales, the local character set is usually an 8–bit superset of ASCII.

❖ **Using Character Streams :**

All character stream classes are descended from Reader and Writer. As with byte streams, there are character stream classes that specialise in file I/O : File Reader and File Writer. The Copy Characters example illustrates these classes.

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Copy Characters
{

 public static void main(String[] args) throws IOException
 {

  FileReader inputStream = null;
  FileWriter outputStream = null;

  try
  {
   inputStream = new FileReader("xanadu.txt");
   outputStream = new FileWriter("characteroutput.txt");

   int c;
   while ((c = inputStream.read()) != -1)
   {
    outputStream.write(c);
```

```
      }
     }
     finally
     {
      if(inputStream != null)
      {
        inputStream.close();
      }

      if (outputStream != null)
      {
        outputStream.close();
      }
     }
    }
   }
```



```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_4$ javac CopyCharacters.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_4$ java CopyCharacters
66
65
79
85
10
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_4$
```

Figure 15.6 Output of Program

Copy Characters is very similar to Copy Bytes. The most important difference is that Copy Characters uses File Reader and File Writer for input and output in place of File Input Stream and File Output Stream. Notice that both Copy Bytes and Copy Characters use an int variable to read to and write from. However, in Copy Characters, the int variable holds a character value in its last 16 bits; in Copy Bytes, the int variable holds a byte value in its last 8 bits.

❖ **Character Streams that use Byte Streams :**

Character streams are often "wrappers" for byte streams. In order to perform the physical I/O, the character stream uses the byte stream while the translation between characters and bytes is handled by the character stream. File Reader, for example, uses File Input Stream, while File Writer uses File Output Stream.

There are two general–purpose byte–to–character "bridge" streams: Input Stream Reader and Output Stream Writer. Use them to create character streams when there are no prepackaged character stream classes that meet your needs. The sockets lesson in the networking trail shows how to create character streams from the byte streams provided by socket classes.

❖ **Line–Oriented I/O :**

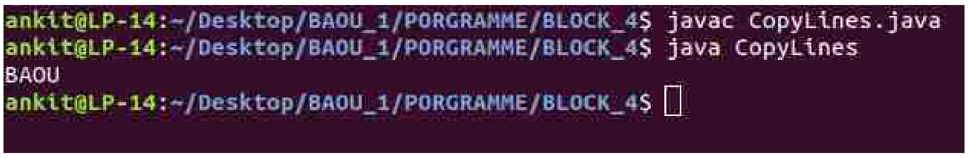
Let us modify the Copy Characters example to use line–oriented I/O. To do this, we have to use two classes we have not seen before, Buffered

Reader and Print Writer. We will explore these classes in greater depth in Buffered I/O and Formatting. Right now, we're just interested in their support for line–oriented I/O.

The Copy Lines example invokes Buffered Reader. Read Line and Print Writer. println to do input and output one line at a time.

```
import java.io. File Reader;
import java.io. File Writer;
import java.io. Buffered Reader;
import java.io. Print Writer;
import java.io.IO Exception;

public class CopyLines
{
 public static void main(String[] args) throws IOException
 {
 BufferedReader inputStream = null
 PrintWriter outputStream = null

 try
 {
  inputStream = new BufferedReader(new FileReader("xanadu.txt"))
 outputStream = new PrintWriter(new FileWriter("characteroutput.txt"))

  while((l = inputStream.readLine()) != null)
  {
   output Stream. println(l)
  }

 }
 finally
 {
  if(inputStream != null)
  {
   inputStream.close()
  }

  if(outputStream != null)
  {
   outputStream.close()
  }
```

```
            }
        }
    }
```



```
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_4$ javac CopyLines.java
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_4$ java CopyLines
BAOU
ankit@LP-14:~/Desktop/BAOU_1/PORGRAMME/BLOCK_4$ 
```

Figure 15.7 Output of Program

Invoking read Line returns a line of text with the line. With the use of println Copy Lines outputs each line, which appends the line terminator for the current operating system. This might not be the same line terminator that was used in the input file.

There are many ways to structure text input and output beyond characters and lines. For more information, see scanning and Formatting.

❖ **Buffered Streams :**

Most of the examples use unbuffered I/O. This means the underlying OS handles directly each read or write request. The result is that the program is made much less efficient as each such request often triggers disk access, network activity, or some other operation that is relatively expensive.

The Java platform implements buffered I/O streams in order to reduce this kind of overhead. Buffered input streams read data from a memory area known as a buffer; the native input API is called only when the buffer is empty. Similarly, buffered output streams write data to a buffer and the native output API is called only when the buffer is full.

There are four buffered stream classes, which can be used to wrap unbuffered streams: Buffered Input Stream and Buffered Output Stream create buffered byte streams, while Buffered Reader and Buffered Writer create buffered character streams.

❖ **Flushing Buffered Streams :**

It often makes sense to write out a buffer at critical points, without waiting for it to fill. This is known as flushing the buffer.

Some buffered output classes support autoflush, specified by an optional constructor argument. Certain key events cause the buffer to be flushed when autoflush is enabled. For example, an autoflush Print Writer object flushes the buffer on every invocation of println or format. In order to flush a stream manually, invoke its flush method. The flush method is valid on any output stream but has no effect unless the stream is buffered.

❖ **Scanning and Formatting :**

To assist you with programming I/O that often involves translating to and from the neatly formatted data humans like to work with, the Java platform provides two APIs. The scanner API breaks input into individual tokens associated with bits of data and the formatting API assembles data into nicely formatted, human–readable form.

❖ **I/O from the Command Line :**

A program is often run from the command line and interacts with the user in the command line environment. The Java platform supports this kind of interaction in two ways: through the Standard Streams and through the Console.

❖ **Standard Streams :**

Standard Streams are a feature of many operating systems. By default, they read input from the keyboard and write output to the display. They also support I/O on files and between programs but that feature is controlled by the command line interpreter, not the program.

The Java platform supports the following three Standard Streams: Standard Input, accessed through System.in; Standard Output, accessed through System.out; and Standard Error, accessed through System.err. These objects are defined automatically and do not need to be opened. Standard Output and Standard Error are both for output; having error output separately allows the user to divert regular output to a file and still be able to read error messages. You might expect the Standard Streams to be character streams, but, for historical reasons, they are byte streams. System.out and System.err are defined as PrintStream objects. Although it is technically a byte stream, PrintStream utilizes an internal character stream object to emulate many of the features of character streams.

By contrast, System.in is a byte stream with no character stream features. To use Standard Input as a character stream, wrap System.in in InputStreamReader.

Input Stream Reader cin = new InputStreamReader(System.in);

The java.io package contains nearly every class you might ever need to perform input and output (I/O) in Java. All these streams represent an input source and an output destination. The stream in the java.io package supports many data such as primitives, Object, localized characters etc.

A stream can be defined as a sequence of data. The InputStream is used to read data from a source and the OutputStream is used for writing data to a destination.

Java does provide strong, flexible support for I/O as it relates to files and networks but this tutorial covers very basic functionality related to streams and I/O. We would see most commonly used example one by one:

❑ **Check Your Progress – 2 :**

1. Explain standard streams.
2. Write a program to illustrate the use of byte streams.

....

....

....

....

....

15.4 Reading Console Input :

Java input console is accomplished by reading from System.in. To obtain a character–based stream that is attached to the console, you wrap System.in in a Buffered Reader object, to create a character stream. Here is most common syntax to obtain Buffered Reader:

BufferedReader br = new BufferedReader(new InputStreamReader (System.in));

Once BufferedReader is obtained, we can use read( ) method to reach a character or read Line( ) method to read a string from the console.

Reading Characters from Console

Now let us learn how to read a character from a BufferedReader, we would use read( ) method whose sytax is as follows:

int read( ) throws IOException

Every time that read ( ) is called, then it reads a character from the input stream and then returns it as an integer value. It returns .1 as soon as the end of the stream is encountered. As you can see, it can throw an IOException.

Reading Strings from Console

Now let us see to read a string from the keyboard, first use the version of readLine( ) which is a member of the BufferedReader class. Its general form is shown here:

String readLine( ) throws IOException

Now the time to understand the following program which demonstrates BufferedReader and the readLine( ) method. This program reads and displays lines of text until you enter the word "end":

```
// Read a string from console using a BufferedReader.
import java.io.*
class BRReadLines {
public static void main(String args[]) throws IOException
   {
      // Create a BufferedReader using System.in
      BufferedReader br = new BufferedReader(new
      InputStreamReader(System.in))
      String str
      System.out.println("Enter lines of
      text.") System.out.println("Enter 'end' to
      quit.") do {
      str = br.readLine() System.out.println(str)
      } while(!str.equals("end"));
   }
}
```

Here is a sample run :

```
Enter lines of text
Enter 'end' to quit
This is line one
This is line one
This is line two
This is line two
end
end
```

❑ **Check Your Progress – 3 :**

1. Write a program to read characters from console.
2. Write the syntax to obtain Buffered Reader

....

....

....

....

....

15.5 Writing Console Output :

To write Console output which is most easily accomplished (done) with print( ) and println( ). These methods are defined by the class Print Stream that is the type of the object referenced by System.out. Even if System.out is a byte stream, using it for simple program output is still acceptable.

Because Print Stream is an output stream resultant(derived) from Output Stream, which also implements the low–level method write(). hence, write() can be used to write to the console. The easiest form of write( ) defined by Print Stream is shown as bellow here:

```
void write(int byteval)
```

This method writes to the stream the byte specified by byteval. Although byteval is declared as an integer, only the low–order eight bits are written.

Example :

Here is a short example that uses write( ) to output the character "A" followed by a newline to the screen:

```
import java.io.*;

//Demonstrate System.out.write().
class WriteDemo
{
 public static void main(String args[])
 {
  int b;
  b = 'A';
  System.out.write(b);
  System.out.write('\n');
 }
}
```

This would produce simply 'A' character on the output screen.

Note: You will not often use write( ) to perform console output because print( ) and println( ) are substantially easier to use.

❑ **Check Your Progress – 4 :**

1. Write the general form of write() defined by Print Stream.
2. How is console output accomplished ?

....

....

....

....

....

15.6 Reading and Writing Files :

Now we explain about, a stream can be defined as a sequence(string) of data. The InputStream which is used to read data from a source and the OutputStream is then used for writing data to a destination (output).

Here, is a hierarchy of classes to deal with Input and Output streams.

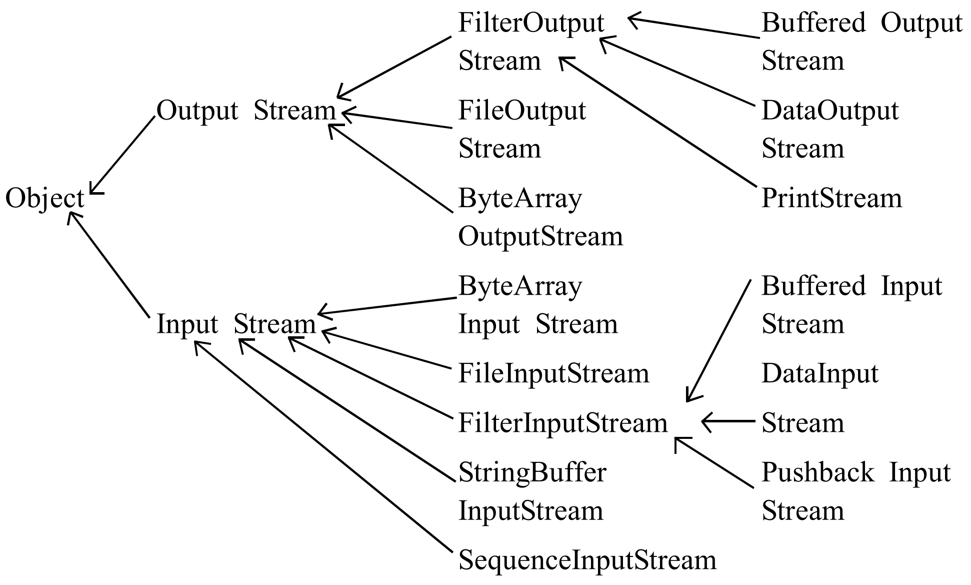


Figure 15.8 Hierarchy of classes to deal with Input and Output Streams

The two important streams are File Input Stream and File Output Stream which would be discussed in this section:

❖ **File Input Stream :**

This stream is used for reading data from the files. Objects can be created using the keyword new and there are several types of constructors available.

Following constructor takes a file name as a string to create an input stream object to read the file:

Input Stream f = new File Input Stream("C:/java/hello");

Following constructor takes a file object to create an input stream object to read the file. First we create a file object using File() method as follows:

File f = new File("C:/java/hello");

Input Stream f = new File InputStream()

Once you have Input Stream object in hand then there is a list of helper methods which can be used to read to stream or to do other operations on the stream.

Table 15.1 : List of helper methods

SN	Methods with Description
1.	public void close() throws IOException{} This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException.
2.	protected void finalise() throws IOException {} This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException.
3.	public int read(int r) throws IOException{} This method reads the specified byte of data from the InputStream. Returns an int. Returns the next byte of data and –1 will be returned if it's end of file.
4.	public int read(byte[] r) throws IOException{} This method reads r.length bytes from the input stream into an array. Returns the total number of bytes read. If end of file –1 will be returned.
5.	public int available() throws IOException{} Gives the number of bytes that can be read from this file input stream. Returns an int.

There are other important input streams available :

- Byte Array Input Stream
- Data Input Stream

❖ **File Output Stream :**

File Output Stream is used to create a file and write data into it. The stream would create a file, if it does not already exist, before opening it for output.

Here are two constructors that can be used to create a File Output Stream object.

Following constructor takes a file name as a string to create an input stream object to write the file :

Output Stream f = new File Output Stream("C:/java/hello")

Following constructor takes a file object to create an output stream object to write the file. First we create a file object using File() method as follows:

File f = new File("C:/java/hello")

Output Stream f = new File Output Stream(f)

Once you have Output Stream object in hand then there is a list of helper methods which can be used to write to stream or to do other operations on the stream.

SN	Methods with Description
1.	public void close() throws IOException{} This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException.
2.	protected void finalise() throws IOException {} This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException.
3.	public void write(int w) throws IOException{} This methods writes the specified byte to the output stream.
4.	public void write(byte[] w) Writes w.length bytes from the mentioned byte array to the OutputStream.

There are other important output streams available :

- Byte Array Output Stream
- Data Output Stream

Example :

Following is the example to demonstrate Input Stream and Output Stream:

```
import java.io.*;

public class file StreamTest
{
 public static void main(String args[])
 {

  try
  {

   byte bWrite [] = {11,21,3,40,5};
   Output Stream os = new File Output Stream("C:/test.txt");
   for(int x=0; x < bWrite.length ; x++)
   {
    os.write( bWrite[x] ); // writes the bytes
   }
   os.close();
    InputStream is = new File Input Stream("C:/test.txt"); int size =
is.available();

   for(int i=0; i< size; i++)
   {
    System.out.print((char)is.read() + " ");
    }

    is.close();
   }
   catch(IOException e)
   {
    System.out.print("Exception");
   }

  }
 }
```

The above code would create file test.txt and would write given numbers in binary format. Same would be output on the std out screen.

❖ **File Navigation and I/O :**

There are several other classes that we would be going through to get to know the basics of File Navigation and I/O.

- File Class
- File Reader Class
- File Writer Class

❖ **Creating Directories in Java :**

There are two useful File utility methods which can be used to create directories :

- The mkdir( ) method creates a directory, returning true on success and false on failure. Failure indicates that the path specified in the File object already exists, or that the directory cannot be created because the entire path does not exist yet.
- The mkdirs() method creates both a directory and all the parents of the directory.

Following example creates "/tmp/user/java/bin" directory:

```
import java.io.File;

class CreateDir
{
 public static void main(String args[])
 {
  String dirname = "/tmp/user/java/bin";
  File d = new File(dirname);
  //Create directory now. d.mkdirs();
 }
}
```

Compile and execute above code to create "/tmp/user/java/bin".

Note : Java automatically takes care of path separators on UNIX and Windows as per conventions. If you use a forward slash (/) on a Windows version of Java, the path will still resolve correctly.

❖ **Reading Directories :**

A directory is a File that contains a list of other files and directories. When you create a File object and it is a directory, the isDirectory( ) method will return true.

You can call list( ) on that object to extract the list of other files and directories inside. The program shown here illustrates how to use list( ) to examine the contents of a directory:

```
import java.io.File

class DirList {
public static void main(String args[]) {
String dirname = "/java"
File f1 = new File(dirname)
if (f1.isDirectory()) {
System.out.println( "Directory of " + dirname)
String s[] = f1.list()
for (int i=0; i < s.length; i++) {
File f = new File(dirname + "/" + s[i])
if (f.isDirectory()) {
System.out.println(s[i] + " is a directory")
} else {
System.out.println(s[i] + " is a file")
}
}
} else {
System.out.println(dirname + " is not a directory")
      }
   }
}
```

This would produce following result :

```
Directory of /mysql
bin is a directory
lib is a directory
demo is a directory
test.txt is a file
README is a file
index.html is a file
include is a directory
```

❑ **Check Your Progress – 5 :**

1. Write the hierarchy of classes to deal with Input and Output streams.
2. Write a note on File Input Stream.

....

....

....

....

....

15.7 Serialization :

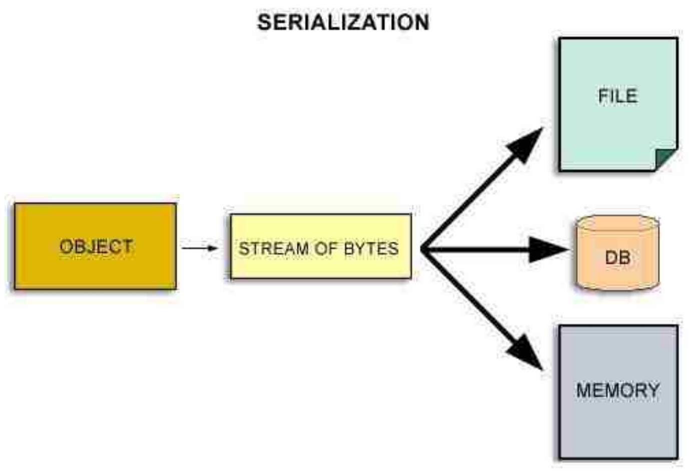


Figure 15.9 Serialization

Let us understand that Java provides a mechanism, called object serialization where an object can be represented (shown) as a sequence(string) of bytes that includes the object's data and also information about the object's type and the types of data stored in the object.

After a serialised object has been written into a file, it can be read from the file and deserialised, i.e., the type information and bytes that represent the object and its data can be used to recreate the object in memory.

Most impressive is that the entire process is JVM independent, meaning an object can be serialised on one platform and deserialised on an entirely different platform.

Classes ObjectInputStream and ObjectOutputStream are high–level streams that contain the methods for serialising and deserialising an object.

The ObjectOutputStream class contains many write methods for writing various data types but one method in particular stands out :

public final void writeObject(Object x) throws IOException

The above method serialises an Object and sends it to the output stream. Similarly, the ObjectInputStream class contains the following method for deserialising an object:

public final Object readObject() throws IOException, ClassNotFound Exception

This method retrieves the next Object out of the stream and deserializes it.

The return value is Object, so you will need to cast it to its appropriate data type.

To demonstrate how serialization works in Java, I am going to use the Employee class that we discussed early on in the book. Suppose that we have the following Employee class, which implements the Serializable interface

```
public class Employee implements java.io.Serialisable
{

 public String name;
 public String address;
 public int transient SSN;
 public int number;

 public void mailCheck()
 {
  System.out.println("Mailing a check to " + name+ " " + address);
 }
}
```

Notice that for a class to be serialised successfully, two conditions must be met:

- The class must implement the java.io.Serialisable interface.
- All of the fields in the class must be serialisable. If a field is not serialisable, it must be marked transient.

If you are curious to know if a Java Satandard Class is Serializable or not, check the documentation for the class. The test is simple: If the class implements java.io.Serialisable, then it is serialisable; otherwise, it is not.

❖ **Serializing an Object :**

The Object Output Stream class is used to serialize an Object.

When the program is done executing, a file named employee.ser is created. The program does not generate any output but study the code and try to determine what the program is doing.

Note : When serialising an object to a file, the standard convention in Java is to give the file a .ser extension.

❖ **Deserialization an Object :**

The following Deserialise Demo program deserialises the Employee object created in the Serialise Demo program. Study the program and try to determine its output:

```
import java.io.*;

public class DeserialiseDemo
{
 public static void main(String [] args)
 {
  Employee e = null;
  try
  {
   FileInputStream fileIn = new FileInputStream("employee.ser");
   ObjectInputStream in = new ObjectInputStream(fileIn);
   e = (Employee) in.readObject();

   in.close();
   fileIn.close();
  }
  catch(IOException i)
  {
   i.printStackTrace();
   return;
  }
  catch(ClassNotFoundException c)
  {
   System.out.println(.Employee class not found.);
   c.printStackTrace();
   return;
  }

  System.out.println("Deserialized Employee...");
  System.out.println("Name: " + e.name);
  System.out.println("Address: " + e.address);
  System.out.println("SSN: " + e.SSN);
  System.out.println("Number: " + e.number);
 }
}
```

This would produce following result :

```
Deserialized Employee...
Name:Reyan Ali
Address:Phokka Kuan, Ambehta Peer
SSN:0
Number:101
```

❑ **Check Your Progress – 6 :**

1. Which are the streams that contain methods for serialising and deserialising an object ?

2. What do you mean by serialisation ?

   ....

   ....

   ....

   ....

   ....

3. The Byte streams are used by programs for the purpose of performing input and output of 8–bit bytes.

   (A) True (B) False

4. To read or write the file we need to import ________ package.

   (A) java.utill (B) java.awt (C) java.net (D) Java.io

5. The character values are stored by Java platformusing ________ conventions. .

   (A) ASCII (B) Unicode

6. A stream can be defined as a sequence of data.

   (A) True (B) False

7. Java input console is accomplished by reading from ________.

   (A) System.input (B) System.take (C) System.out (D) System.in

8. To write Console output ________ method is used.

   (A) out() (B) pritnf() (C) cout() (D) println()

9. The InputStream class is an abstract class.

   (A) True (B) False

10. The Print Stream class is a subclass of ________.

    (A) IO (B) Stream

    (C) Filter output Stream (D) Filter input Stream

11. ________ method is used to read single charater from file.

    (A) take() (B) readdata() (C) readchar() (D) read()

12. To get the detail about file ________ class is used.

    (A) FileInfo (B) FileReader (C) Reader (D) File

15.8 Let Us Sum Up :

This Unit No. 13 has got importance because details regarding the files and its Management like Input and output. Java I/O is said as Java Input/ Output and is a part of java.io package. Now package has a Input Stream and Output Stream. The Java Input Stream is meant for reading the stream, byte stream and array of byte stream. This can be used for memory allocation. The Output Stream is used for writing byte and array of bytes.

An input source or an output target is represented (shown) by an I/O Stream. A stream can represent disk files, devices, other programs and memory arrays and also as many different kinds of sources and destinations target. BYTE STREAMS: The Byte streams are used by programs for the purpose of performing input and output of 8–bit bytes. Here all byte stream classes are descended from Input Stream and Output Stream. The character values are stored by Java platformusing

Unicode conventions. Character stream I/O automatically translates this internal format to and from the local character set. In Western locales, the local character set is usually an 8–bit superset of ASCII.

There is learning related to use line–oriented I/O, unbuffered I/O, flushing the buffer, Scanning and formatting and standard stream. Next thing came in learning related to read Console. To write Console output which is most easily accomplished (done) with print( ) and println( ). These methods are defined by the class Print Stream that is the type of the object referenced by System.out. Even even if System.out is a byte stream, using it for simple program output is still acceptable.Now futher we explain how to read and write files.

The Input Stream which is used to read data from a source and the Output Stream is then used for writing data to a destination (output).

Serialization where an object can be represented as a sequence (string) of bytes that includes the object's data and also information about the object's type and the types of data stored in the object. Next we have understood about Serializing an Object which is nothing but the Object Output Stream class is used to serialize an Object and Deserialising an Object.

15.9 Suggested Answer for Check Your Progress :

- **Check Your Progress 1 :**
  See Section 15.2
- **Check Your Progress 2 :**
  See Section 15.3
- **Check Your Progress 3 :**
  See Section 15.4
- **Check Your Progress 4 :**
  See Section 15.5
- **Check Your Progress 5 :**
  See Section 15.6
- **Check Your Progress 6 :**
  See Section 15.7
- **Check Your Progress 7 :**

1 : See Section 15.8		**2 :** See Section 15.8		**3 :** A
4 : D	**5 :** B	**6 :** A	**7 :** D	**8 :** D
9 : A	**10 :** C	**11 :** D	**12 :** D	

15.10 Glossary :

1. **Stream –** A stream can represent disk files, devices, other programs and memory arrays and also as many different kinds of sources and destinations target.

2. **Byte streams** – are used by programs for the purpose of performing input and output of 8–bit bytes.
3. **Standard Streams** – Standard Streams are a feature of many operating systems. By default, they read input from the keyboard and write output to the display. They also support I/O on files and between programs but that feature is controlled by the command line interpreter, not the program.
4. **Serialization** – where an object can be represented (shown) as a sequence(string) of bytes that includes the object's data and also information about the object's type and the types of data stored in the object

15.11 Assignment :

Write a program to create a file student to store the roll no., name and marks of 3 subjects of 5 students using the features of Java.

15.12 Activities :

1. Explain how to create directories in Java.
2. Explain the methods that can be used to read to stream.

15.13 Case Study :

Read a file of sums and products (one per line) and write the values of the expressions to another file. The two file names are expected to be given as command–line arguments

15.14 Further Reading :

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000
2. Java 2, The Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999
3. Programming with Java, Ed. 2,E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000
4. The Java Tutorial, Ed. 2, 2 volumes, Mary Campione and Kathy Walrath, Addison Wesley Longmans, 1998
5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele & Gilad Bracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000
6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition, 2000

BLOCK SUMMARY :

Finally lot of learning took place after reading complete block

First Unit of this Block made us learn about Abstract Window Toolkit which is nothing but is Java's original platform–independent windowing, graphics and user–interface widget toolkit. The AWT is now part of the Java Foundation Classes (JFC) the standard API for providing a graphical user interface (GUI) for a Java program. A Window provides a top–level window on the screen, with no borders or menu bar. It provides a way to implement pop–up messages, among other things. The default layout for a Window is Border Layout.

Working with Frames – A Frame is a Window with all the window manager's adornments (window title, borders, window minimize/maximize/close functionality) added. It may also include a menu bar. Since Frame subclasses Window, its default layout is Border Layout. Frame provides the basic building block for screen–oriented applications. As long as working with Graphics is concern Java provides numerous primitives for drawing lines, squares, circles, polygons and images. The figure shows a simple drawing. The Font, Font Metrics, Color and System Color classes provide the ability to alter the displayed output. With the Font class, you adjust how displayed text will appear. With Font Metrics, you can find out how large the output will be, for the specific system the user is using. You could use the Color class to set the color of text and graphics.

We have learned about Controls where in Labels, Buttons, Check Boxes, Radio Button, Text Area, and Text Field covered. After this we learned about Layout Manager. In this All Containers, by default, has a layout manager; an object that implements the Layout Manager interface. If a Container's default layout manager doesn't go well with your needs, you can easily replace it with another one. We further came to know about Rule those can be said as you have to openly tell a container not to use a layout manager, it is linked with its own occasion of a layout manager. This layout manager is automatically/ by default consulted every time the Container might need to change its appearance. Most layout managers do not require programs to directly call their respective methods.

We also seen and understood about rules for Layout Manager like Flow layout, Grid Layout and Grid bag layout. Further we have covered event handling. Events are the integral part of the java platform. You can see the concepts related to the event handling through the example and use methods through which you can implement the event driven application.

Further we have also learned about there are some event listeners that have multiple methods to implement. That is some of the listener interfaces contain more than one method. For instance, the Mouse Listener interface contains five methods such as mouse Clicked, mouse Pressed, mouse Released etc. There is also mention about Inner classes cannot have static members, only static final variables. 1. Interfaces are never inner. 2. Static classes are not inner classes. We have also learned about anonymous inner classes. There is in detail learning about applets, applets life cycle.

Now the Unit no. 13 has got importance because details regarding the files and it's Management like Input and output. Java I/O is said as Java Input/ Output and is a part of java.io package. Now package has an Input Stream and Output Stream. The Java Input Stream is meant for reading the stream, byte stream and array of byte stream. This can be used for memory allocation. The Output Stream is used for writing byte and array of bytes.

An input source or an output target is represented (shown) by an I/O Stream. A stream can represent disk files, devices, other programs and memory arrays and also as many different kinds of sources and destinations target. BYTE STREAMS : The Byte streams are used by programs for the purpose of performing input and output of 8–bit bytes. Here all byte stream classes are descended from Input Stream and Output Stream. The character values are stored by Java platformusing Unicode conventions. Character stream I/O automatically translates this internal format to and from the local character set. In Western locales, the local character set is usually an 8–bit superset of ASCII. There is learning related to use line–oriented I/O, unbuffered I/O, flushing the buffer, Scanning and formatting and standard stream. Next thing came in learning related to read Console. To write Console output which is most easily accomplished (done) with print( ) and println( ). These methods are defined by the class Print Stream that is the type of the object referenced by System.out. Even even if System.out is a byte stream, using it for simple program output is still acceptable.Now futher we explain how to read and write files. The Input Stream which is used to read data from a source and the Output Stream is then used for writing data to a destination (output).

Serialization where an object can be represented as a sequence (string) of bytes that includes the object's data and also information about the object's type and the types of data stored in the object. Next we have understood about Serializing an Object which is nothing but the Object Output Stream class is used to serialize an Object and Deserialising an Object.

BLOCK ASSIGNMENT :

❖ **Short Questions :**

1. Why graphics is considered an abstract class ?
2. Give the use of canvas and container objects.
3. Write a note on buttons.
4. Write down the procedure for event handling.
5. What are adapter classes ?
6. Give the use of adapter classes
7. Write a note on inner classes.
8. What could be declared a local or anonymous class ?
9. Define an anonymous inner class.

❖ **Long Questions :**

1 Explain the text area component of AWT package

2 Give the general rules for using Layout Manager.

3 Explain Card Layout

4 Explain semantic events

5 Explain how thread itself is made into an anonymous inner class

6 Give the structure of applets.

7 Write a note on the other applet methods

8 Write the hierarchy of classes to deal with Input and Output streams.

9 Write a note on File Input Stream

BIBLIOGRAPHY

http://docstore.mik.ua/orelly/java/awt/ch01_05.htm http://www.oreilly.com/openbook/javawt/book/ch01.pdf

http://www.it.cas.cz/manual/java/uiswing/layout/border.html

http://journals.ecs.soton.ac.uk/java/tutorial/ui/layout/flow.html

http://da2i.univ–lille1.fr/doc/tutorial–java/uiswing/layout/flow.html

http://www.eg.bucknell.edu/~mead/Java–tutorial/ui/swingLayout/grid.html

http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

http://www.roseindia.net/java/example/java/awt/event–handling.shtml

http://www.javabeat.net/inner–classes–in–java/

http://readwall.wap.blog.163.com/w2/blogDetail.do;jsessionid=A378CA75233AD84B146C854CF4DB0D16.blog57–8010?blogId=fks_080068087084088066082083086095085086080065083087085065&showRest=true&p=2&hostID=readwall

http://www.hostitwise.com/java/japplet.html

http://www.studiesinn.com/learn/Programming–Languages/Java–Language/Running–an–Applet.html

http://stanwir.seecs.nust.edu.pk/Lectures/java/IOStreams.pdf

http://www.aliftutorials.com/javase/streams/scanfor.html

http://espirit.in/simplified/java/intermediate/stream.html

❖ **Enrolment No. :** ☐

1. How many hours did you need for studying the units ?

Unit No.	12	13	14	15
No. of Hrs.				

2. Please give your reactions to the following items based on your reading of the block :

Items	Excellent	Very Good	Good	Poor	Give specific example if any
Presentation Quality	☐	☐	☐	☐	
Language and Style	☐	☐	☐	☐	
Illustration used (Diagram, tables etc)	☐	☐	☐	☐	
Conceptual Clarity	☐	☐	☐	☐	
Check your progress Quest	☐	☐	☐	☐	
Feed back to CYP Question	☐	☐	☐	☐	

3. Any other Comments

....

....

....

....

....

....

....

....