BAOU Education for All

Dr. Babasaheb Ambedkar Open University

(Established by Government of Gujarat)

NAAC ACCREDITED A++ GRADE 3.55/4

Web Development Technology using Java

BSCCS-504

Bachelor Of Science (Hons.)
Cyber Security
(BSCCS)

2024

Web Development Technology using Java

Dr. Babasaheb Ambedkar Open University

Web Development Technology using Java

Expert Committee

Prof. (Dr.) Nilesh K. Modi Professor and Director, School of Computer Science, Dr. Babasaheb Ambedkar Open University, Ahmedabad	(Chairman)
Prof. (Dr.) Ajay Parikh Professor and Head, Department of Computer Science Gujarat Vidyapith, Ahmedabad	(Member)
Prof. (Dr.) Satyen Parikh Dean, School of Computer Science and Application Ganpat University, Kherva, Mahesana	(Member)
M. T. Savaliya Associate Professor and Head Computer Engineering Department Vishwakarma Engineering College, Ahmedabad	(Member)
Mr. Nilesh Bokhani Assistant Professor, School of Computer Science, Dr. Babasaheb Ambedkar Open University, Ahmedabad	(Member)
Dr. Himanshu Patel Assistant Professor, School of Computer Science, Dr. Babasaheb Ambedkar Open University, Ahmedabad	(Member Secretary)

Course Writer

Dr. Vinod L Desai — Associate Professor,
Department of Computer Science and Technology,
Sardar Patel University,
Vallabh Vidyanagar - Gujarat

Content Reviewer

Prof. (Dr.) Nilesh K. Modi — Professor and Director,
School of Computer Science,
Dr. Babasaheb Ambedkar Open University,
Ahmedabad - Gujarat

Content Editor

Mr. Nilesh Bokhani — Assistant Professor,
School of Computer Science,
Dr. Babasaheb Ambedkar Open University,
Ahmedabad - Gujarat

ISBN -

Printed and published by: Dr. Babasaheb Ambedkar Open University, Ahmedabad While all efforts have been made by editors to check accuracy of the content, the representation of facts, principles, descriptions and methods are that of the respective module writer. Views expressed in the publication are that of the authors, and do not necessarily reflect the views of Dr. Babasaheb Ambedkar Open University.

Dr. Babasaheb Ambedkar Open University

BSCCS-504

Web Development Technology using Java

BLOCK-1: Swing, Event Handling and Networking

BLOCK-2: JDBC, STORED PROCEDURE AND FUNCTIONS

BLOCK-3: Web APPLICATION, SERVLETS AND SESSION MANAGEMENT

BLOCK-4: JSP, EXPRESSION LANGUAGE AND JSTL

BLOCK 1: Swing, Event Handling and Networking

Block Introduction

Through Swing user can design a GUI application as per their requirements. Swing supports feature rich components to design the GUI application. It also has various event handling classes and interfaces to deal with.

Through Java Networking two or more computing devices or machines together can communicate and share resources. Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

In this block, we will cover in detail about the Basics of GUI programming using Swing components, its superiority on AWT, event handling with swing components and arranging these components on the containers with the help of Layout Manger. We will also discuss the fundamentals of Networking, different kinds of Protocols and difference between TCP and UDP. The block also focuses on the study and understanding of networking classes of network package available in JDK. The students will be given an idea on creating Client and Server program with the help of Sockets programs. The concept related to working and implementing of connection oriented and connection-less client-server programs, multicasting the data are also explained practically.

Block Objective

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

- Importance of GUI application
- Designing desktop GUI application
- The use of different controls to accept input
- Designing interactive GUI application
- Defining event handlers for various kinds of events
- To arrange component on containers using layout manager
- about Networking Protocols
- Difference between TCP and UDP
- Different Networking Classes available in JDK
- Concept of Sockets, IP Address and Port numbers in Networking
- Qualities of creating Client and Server Program using Sockets
- Qualities of creating Client and Server Program using DatagramSocket & DatagramPacket
- Qualities of creating Client and Server Program using MulticastSockets
- Idea about running Client Server Programs

Block Structure

Unit 1:	GUI Programming using Swing
Unit 2:	Event handling and Layout
Unit 3:	Java Networking
Unit 4:	Java.net Package

Block Summary

In this block, students have learnt and understand about GUI programming using SWING. They also learnt designing and developing event driven programming using Delegation Event Model. They learnt that java networking helps to connect two or more machines to make communication. This block has explored various functionality of java.awt, javax.swing, java.awt.event and java.net class library of JDK. The students were well explained the concepts of creating and designing GUI application along with event listeners and event handlers. The concept related to TCP, UDP, Socket, ServerSocket along with INetAddress and MulticastSocket were discussed with its applications.

Block Assignment

Short Answer Questions:

1. Discuss the importance of GUI application.
2. “Swing components are lightweight”. Justify.
3. Define event.
4. Write short note on INetAddress class.
5. Define URI and URL.
6. Which method does MouseMotionListener handles?

Long Answer Questions:

1. Discuss different steps required for connecting client and server using TCP.
2. Write short notes on URLConnection.
3. Write short note on MulticastSocket.
4. Write note on Event Delegation Model.
5. Explain various methods of Key Listener.
6. Dicuss about URLClassLoader, URLDecoder, URLEncoder and URLStreamHandler.

UNIT 1: GUI PROGRAMMING USING SWING

Unit Structure

1.0 Learning Objectives

After learning this Unit, you will be:

- Able to design desktop GUI application
- Able to use different controls to accept input

1.1 Introduction

GUI stands for Graphical User Interface, is a user-friendly visual display for Java applications. It consists of graphical controls like buttons, labels, containers etc. through which users can interact with an application. A GUI consists of an array of user interface elements which are displayed when a user is interacting with an application. Swing was designed with a dynamic architecture to make the elements customizable and easy to render as plug-and-play. Swing is a lightweight GUI toolkit. It is a part of the JFC (Java Foundation Classes). It is build on top of the AWT API and purely written in java. It is platform independent unlike AWT.

There are generally three Java APIs used for graphics programming in java like AWT (Abstract Windowing Toolkit), Swing and JavaFX.

I. AWT API was introduced in JDK 1.0. Many of the AWT UI components have become obsolete and replaced by newer Swing UI components.

II. Swing API extends the AWT, was introduced as part of Java Foundation Classes (JFC) after the release of JDK 1.1. JFC consists of Swing, Accessibility, Java2D, Pluggable Look-and-Feel and Internationalization APIs. JFC was integrated into core Java since JDK 1.2.

III. The JavaFX is integrated into JDK 8. Its main purpose was to replace Swing. JavaFX was moved out from the JDK in JDK 11, but it is still available as a separate module.

1.2 Differences between Swing and AWT

AWT stands for Abstract Window Toolkit. It is a platform-dependent API to develop GUI (Graphical User Interface) or window-based applications in Java. It was developed by Sun Microsystems In 1995. The major differences between AWT and Swing are following.

API Package: The AWT Component classes are contained in the java.awt package while the Swing component classes are contained in the javax.swing package.

Platform dependency: The AWT components are mainly dependent on the operating system. The Swing components are purely written in java hence not dependent on the operating system.

Weightiness: The AWT is heavy weight since it uses the resources of the operating system. The Swing Components are built on the top of AWT and doesn't need any operating system resources for processing. So, The Swing is mostly lightweight.

Appearance: The Appearance of AWT Components is mainly depends on the operating system's look and feels. The Swing Components are mainly support pluggable look and feel.

Number of Components: The Java AWT provides a smaller number of components. Java Swing provides a greater number of components than AWT, like list, scroll panes, tree, spinners, tables, color choosers etc.

Full-Form: Java AWT stands for Abstract Window Toolkit. Java Swing is referred as Java Foundation Classes (JFC).

Memory: Java Swing needs less memory space as compared to Java AWT.

Time: AWT is slower than swing in terms of execution time while Swing is faster than the AWT.

MVC pattern: MVC pattern is not supported by AWT. MVC pattern is supported by Swing.

2D Graphics Rendering: AWT doesn't support it while Swing supports it.

1.3 Swing Containers

Container classes are classes that can contain other components on it. Java Swing Framework contains a large set of these components with rich functionalities and allows high level of customization. They all are derived from JComponent class and are lightweight components. This class provides some common functionality like pluggable look and feel, support for accessibility, drag and drop, layout etc. A container provides a space where a component can be managed and displayed. Containers are of two types:

Top level Containers:

It inherits Component and Container of AWT. It cannot be contained within other containers.

Heavyweight container includes JFrame, JDialog and JApplet.

Lightweight Containers:

It inherits JComponent class. It is a general purpose container. It can be used to organize related components together. JPanel is a Lightweight Container.

Like AWT application, a Swing application requires a top-level container. There are three top-level containers in Swing:

- JFrame: It is used for the application's main window. A Frame provides the main window for GUI application. It has a title bar (containing an icon, a title, the

minimize, maximize/restore-down and close buttons), an optional menu bar and the content display area.

- JDialog: It is a secondary pop-up window used for interacting with the users. A Dialog has a title-bar (containing an icon, a title and a close button) and a content display area.
- JApplet: It is used for the applet's display-area (content-pane) inside a browser's window. JApplet is the top-level container for an applet, a java program running inside a browser. Applet is no longer supported in most of the browsers.

There are secondary containers (such as JPanel) which can be used to group and layout relevant components. It is a rectangular box used to layout a set of related GUI components in pattern such as grid or flow.

The JComponents shall not be added directly onto the top-level container (e.g., JFrame) because they are lightweight components. The JComponents must be added onto the content-pane of the top-level container. Content-pane is in fact a java.awt.Container that can be used to group and layout components.

Check your progress 1

1. Which of the following architecture does the Swing framework use?

a. MVC

b. MVP

c. Layered architecture

d. Master-Slave architecture

2. A ____ is the abstract foundation class for SWING's non-menu user interface controls?

a. Container

b. Jcomponent

c. Component

d. AWT

3. A ____ is the basic class for all SWING UI components?

a. Container

b. Jcomponent

c. Component

d. AWT

4. Which class is the base class for all Swing components?

a. java.awt.Component

b. java.awt.Container

c. javax.swing.JComponent

d. javax.swing.JPanel

5. Which of the following is not a Swing container?

a. JFrame

b. JDialog

c. JButton

d. JTabbedPane

1.4 Basics of Swing Program

To create a GUI in Swing first create a class that represents the main GUI. This class will extend a top container which will hold all the other components to be displayed. In most of the classes, the main container is a frame, i.e., the JFrame class in javax.swing package. A frame is just like a window which is displayed whenever a user opens an application. Frame has a title bar, menu bar and buttons such as minimize, maximize and close along with other features.

The JFrame class has simple constructors such as JFrame() and JFrame(String). The frame's title bar in the JFrame() constructor remains empty, whereas the JFrame(String) constructor places the title bar with a specified text represented by string argument. Apart from the title, the size of the frame can also be customized by implementing the setSize(int, int) method by providing the width and height desired for the frame as an argument in pixels. For example, calling setSize(500,300) would create a frame that would be 500 pixels wide and 300 pixels long. Generally, frames are invisible at the time of their creation. To make them visible a user has to implement the frame's setVisible(boolean) method by passing the value 'true' as an argument. The general template of Swing program is shown below.

Swing Template:

```
import java.awt.*;        // For AWT layouts
import java.awt.event.*;  // For AWT event classes and listener interfaces
import javax.swing.*;     // For Swing components and containers
// A Swing GUI application inherits from top-level container javax.swing.JFrame
```

```
public class SwingTemplate extends JFrame
{
  // Private instance variables
 // ......
 // Constructor to setup the GUI components and event handlers
 public SwingTemplate() {
    // Get the top-level content-pane from JFrame
    Container cp = getContentPane();
     // Content-pane sets layout
    cp.setLayout(new ....Layout());
     // Create the GUI components
    // .....
     // Content-pane adds components
    cp.add(....);
     // Source object adds listener
    // .....
 }
  // The entry main() method
 public static void main(String[] args)
 {
      //create JFrame object with SwingTemplate
      JFrame jframe = new SwingTemplate ();
     // Exit the program when the close-window button clicked
      jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      // sets title of JFrame
      Jframe.setTitle("......");
      //set size of GUI screen
      jframe.setSize(400,400);
      jframe.setVisible(true);
  }
}
```

After creating the GUI with top level container as a frame user can add components like buttons, text fields etc. to the frame.

Check your progress 2

1. Which of the following methods is used to add a component to a container?

a. addComponent()

b. addContainer()

c. addComponentToContainer()

d. add()

2. Which of the following methods is used to set the size of a container?

a. setSize()

b. setPreferredSize()

c. setBounds()

d. setMinimumSize()

3. Which of the following methods is used to set the layout manager of a container?

a. setLayoutManager ()

b. setLayout()

c. setContainerLayout()

d. setComponentLayout()

1.5 Working with Swing Component

To create a component in Java, the user is required to create an object of that component's class. Following are the widely used swing components.

JButton:

One such important component to implement is JButton. The button can include some text or images on it. It yields an event when clicked and double-clicked. To implement a JButton in the application its constructors can be used.

Constructor Syntax:

1. JButton btn = new JButton("ClickMe");

This constructor returns a button with the text ClickMe on it.

2. JButton btn = new JButton(Icon);

This constructor returns a button with a Icon on it.

3. JButton btn = new JButton(Icon, "ClickMe");

This constructor returns a button with the icon and text as "ClickMe".

Example: **SwingComponent.java**

```
import java.awt.*;
import javax.swing.*;
```

```
public class SwingComponent extends JFrame
{
  public SwingComponent()
  {
            Container cp = getContentPane();
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     JButton bt1 = new JButton("Yes");
     JButton bt2 = new JButton("No");
     cp.setLayout(new FlowLayout());
     cp.add(bt1);
     cp.add(bt2);
  }
  public static void main(String[] args)
  {
      JFrame jframe = new SwingComponent ();
      jframe.setTitle("JButton");
      jframe.setSize(250,200);
      jframe.setVisible(true);
   }
}
```

The setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE) method is used Exit the application. After successful compilation, it will have following GUI when we execute above program:

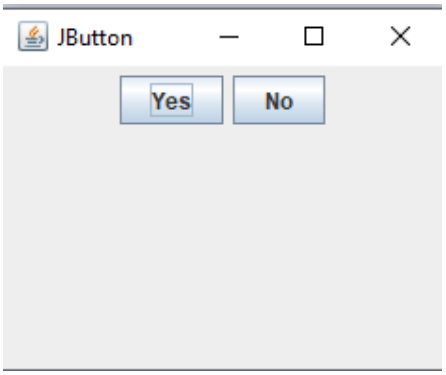


Figure 1. JButton Example

JLabel

JLabel class is used to render a read-only text label or images on the UI. It does not generate any event.
Syntax:
JLabel textLbl = new JLabel("Name:");
This constructor returns a label with specified text.
JLabel imgLbl = new JLabel(Icon);
This constructor returns a label with a specified icon.
The JLabel Contains four constructors. They are as follows:
1. JLabel()
2. JLabel(String str)
3. JLabel(Icon i)
4. JLabel(String str, Icon i, int horizontalAlignment)

JTextField:

JTextField is used for taking input of single line of text from user. It is most widely used text component. It has three constructors,
1. JTextField(int cols)
2. JTextField(String txt, int cols)
3. JTextField(String txt)
In above constructors, cols represent the number of columns in text field and txt represents the default text to be displayed in text field.
Example: **SwingComponent.java**

```
import java.awt.*;
import javax.swing.*;
public class SwingComponent extends JFrame
{
  public SwingComponent()
  {
     Container cp = getContentPane();
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JLabel msg=new JLabel("Message:");  // Label for TextField
     JTextField jtf = new JTextField("Welcome to BAOU",20);
     cp.setLayout(new FlowLayout());
     cp.add(msg);
     cp.add(jtf);
  }
  public static void main(String[] args)
  {
       JFrame jframe = new SwingComponent();
```

```
        jframe.setTitle("JTextField");
        jframe.setSize(320,200);
        jframe.setVisible(true);
    }
}
```

After successful compilation, it will have following GUI when we execute above program:

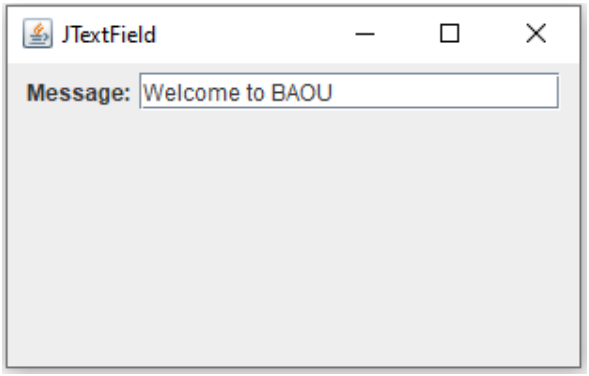


Figure 2. JTextField Example

JCheckBox

The JCheckBox renders a check-box with a label. This control allows user to select multiple items. The check-box has two states, i.e., on and off. On selecting, the state is set to "on," and a small tick is displayed inside the box.

Syntax:

CheckBox chkBox = new JCheckBox(“BAOU”, true);

It returns a checkbox with the label BAOU. Notice the second parameter in the constructor. It is a boolean value indicates the default state of the check-box. True means the default selection i.e. the "on" state.

Example: **SwingComponent.java**

```
import java.awt.*;
import javax.swing.*;
public class SwingComponent extends JFrame
{
  public SwingComponent()
  {
     Container cp = getContentPane();
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JCheckBox jcb = new JCheckBox("BAOU",true);
    JCheckBox jcb1 = new JCheckBox("GTU");
    JCheckBox jcb2 = new JCheckBox("NGU");
```

```
    cp.setLayout(new FlowLayout());
    cp.add(jcb);
    cp.add(jcb1);
    cp.add(jcb2);
 }
 public static void main(String[] args)
 {
      JFrame jframe = new SwingComponent();
      jframe.setTitle("JCheckBox");
      jframe.setSize(320,200);
      jframe.setVisible(true);
  }
}
```

After successful compilation, it will have following GUI when we execute above program:

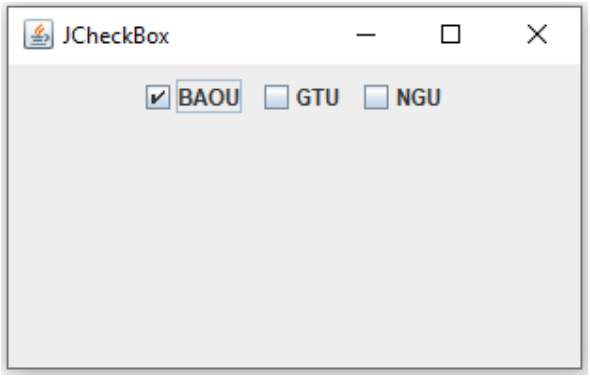


Figure 3. JCheckBox Example

JRadioButton

Radio button is a group of related button in which only one item can be selected. JRadioButton class is used to create a radio button in Frames. It is a group of related buttons from which user can select only one. Users can select one choice from the group. It should be added in ButtonGroup to select only one radio button at a time.

Constructors:

1. JRadioButton(): It creates an unselected radio button with no text.
2. JRadioButton(String str): It creates an unselected radio button with specified text.
3. JRadioButton(String str, boolean state): It creates a radio button with the specified text and selected status.

Example: **SwingComponent.java**

```
import java.awt.*;
import javax.swing.*;
```

```
public class SwingComponent extends JFrame
{
  public SwingComponent()
  {
    Container cp = getContentPane();
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JRadioButton rb1=new JRadioButton("Male");
        JRadioButton rb2=new JRadioButton("Female");
        ButtonGroup bg=new ButtonGroup();
        bg.add(rb1);
        bg.add(rb2);
        cp.setLayout(new FlowLayout());
        cp.add(rb1);
        cp.add(rb2);
  }
 public static void main(String[] args)
  {
      JFrame jframe = new SwingComponent();
      jframe.setTitle("JRadioButton");
      jframe.setSize(320,200);
      jframe.setVisible(true);
   }
}
```

After successful compilation, it will have following GUI when we execute above program:

Figure 4. JRadioButton Example

JComboBox:

Combo box is a combination of text fields and drop-down list. JComboBox is used to show popup menu of items. Item selected by user is shown on the top of a menu.
Constructors:
1. JComboBox(): It creates a JComboBox with a default data model.
2. JComboBox(Object[] items): It creates a JComboBox containing the elements in the specified array.
3. JComboBox(Vector<?> items): It creates a JComboBox containing the elements in the specified Vector.
Example: **SwingComponent.java**

```
import java.awt.*;
import javax.swing.*;
public class SwingComponent extends JFrame
{
  public SwingComponent()
  {
    Container cp = getContentPane();
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        String player[]={"Ved","Het","Kalp","Moksh","Shrey"};
        JComboBox cb=new JComboBox(player);
          cp.setLayout(new FlowLayout());
          cp.add(cb);
 }
 public static void main(String[] args)
 {
     JFrame jframe = new SwingComponent();
     jframe.setTitle("JComboBox");
     jframe.setSize(320,200);
     jframe.setVisible(true);
  }
}
```

After successful compilation, it will have following GUI when we execute above program:

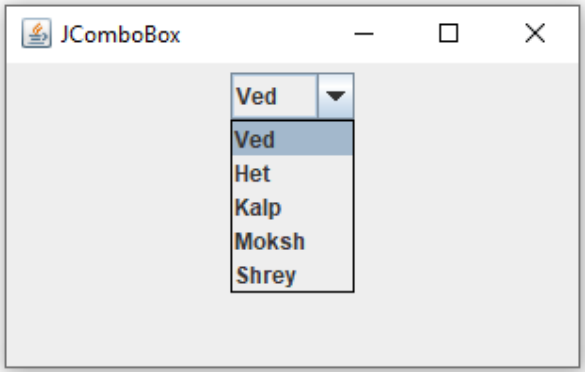


Figure 5. JComboBox Example

JTextArea

A JTextArea Class is used for displaying multiple-line text. It allows editing of multiple line text.

Declaration:

public class JTextArea extends JTextComponent

Constructor:

1. JTextArea(): It creates a text area that displays no text initially.
2. JTextArea(String str): It creates a text area that displays specified text initially.
3. JTextArea(int row, int column): It creates a text area with the specified number of rows and columns that displays no text initially.
4. JTextArea(String str, int row, int column): It creates a text area with the specified number of rows and columns that displays specified text.

Example: **SwingComponent.java**

```
import java.awt.*;
import javax.swing.*;
public class SwingComponent extends JFrame
{
  public SwingComponent()
  {
    Container cp = getContentPane();
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JTextArea jtarea=new JTextArea("Welcome to BAOU");
        cp.setLayout(new FlowLayout());
        cp.add(jtarea);
 }
 public static void main(String[] args)
```

```
 {
      JFrame jframe = new SwingComponent();
      jframe.setTitle("JTextArea");
      jframe.setSize(320,200);
      jframe.setVisible(true);
  }
}
```

After successful compilation, it will have following GUI when we execute above program:

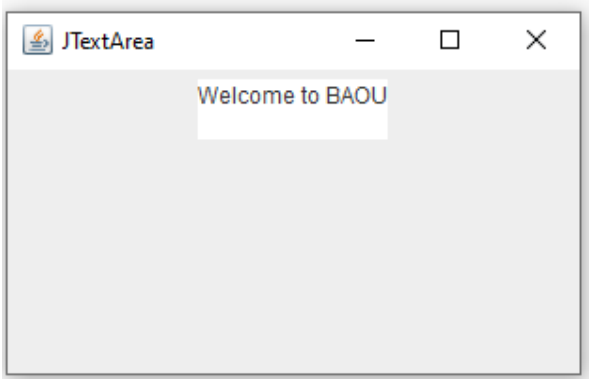


Figure 6. JTextArea Example

JPasswordField:

JPasswordField Class is specifically used for the password and we can edit it. The JPasswordField has 4 constructors:

1. JPasswordField()
2. JPasswordField(int columns)
3. JPasswordField(String txt)
4. JPasswordField(String txt, int columns)

Example: **SwingComponent.java**

```
import java.awt.*;
import javax.swing.*;
public class SwingComponent extends JFrame
{
  public SwingComponent()
  {
     Container cp = getContentPane();
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     JPasswordField jpd = new JPasswordField(20);
    JLabel lbl=new JLabel("Password:");
```

```
        cp.setLayout(new FlowLayout());
        cp.add(lbl);
        cp.add(jpd);
 }
 public static void main(String[] args)
 {
     JFrame jframe = new SwingComponent();
     jframe.setTitle("JPasswordField");
     jframe.setSize(320,200);
     jframe.setVisible(true);
  }
}
```

After successful compilation, it will have following GUI when we execute above program:

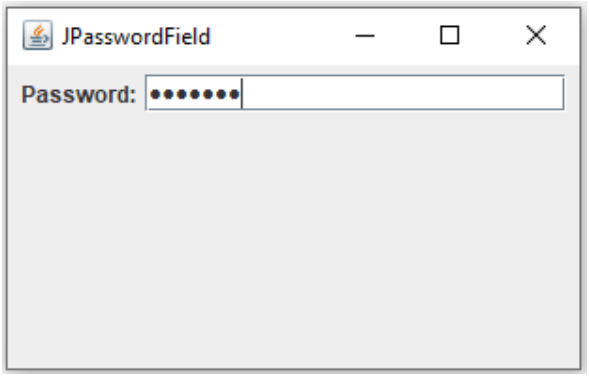


Figure 7. JPasswordField Example

JMenuBar, JMenu and JMenuItem:

The JMenuBar class is used to display menubar on the window or frame. It may contain several menus. The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class. The object of JMenuItem class adds a simple menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

Example: **SwingComponent.java**

```
import java.awt.*;
import javax.swing.*;
public class SwingComponent extends JFrame
{
  public SwingComponent()
  {
```

```
    Container cp = getContentPane();
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JMenu menu;
        JMenuItem op,sv;
        menu = new JMenu("File");
        JMenuBar jm = new JMenuBar();
        op = new JMenuItem("Open");
        sv = new JMenuItem("Save");
        menu.add(op);
        menu.add(sv);
        jm.add(menu);
        setJMenuBar(jm);
        cp.setLayout(new FlowLayout());
 }
 public static void main(String[] args)
 {
     JFrame jframe = new SwingComponent();
     jframe.setTitle("JMenu");
     jframe.setSize(320,200);
     jframe.setVisible(true);
  }
}
```

After successful compilation, it will have following GUI when we execute above program:

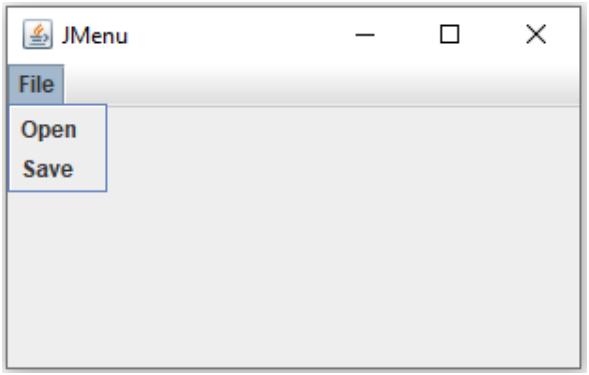


Figure 8. JMenu Example

JScrollBar :

In Java, Swing toolkit contains a JScrollBar class. It is under package javax.swing.JScrollBar class. It is used for adding horizontal and vertical scrollbar.
Constructors:
1. JScrollBar()
2. JScrollBar(int orientation)
3. JScrollBar(int orientation, int value, int extent, int min, int max)
Example: **SwingComponent.java**

```
import java.awt.*;
import javax.swing.*;
public class SwingComponent extends JFrame
{
  public SwingComponent()
  {
     Container cp = getContentPane();
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     JScrollBar hjsBar=new JScrollBar(JScrollBar.HORIZONTAL);
     JScrollBar vjsBar=new JScrollBar(JScrollBar.VERTICAL);
          cp.setLayout(new FlowLayout());
          cp.add(hjsBar);
          cp.add(vjsBar);
  }
  public static void main(String[] args)
  {
       JFrame jframe = new SwingComponent();
       jframe.setTitle("JScrollBar");
       jframe.setSize(320,200);
       jframe.setVisible(true);
   }
}
```

After successful compilation, it will have following GUI when we execute above program:

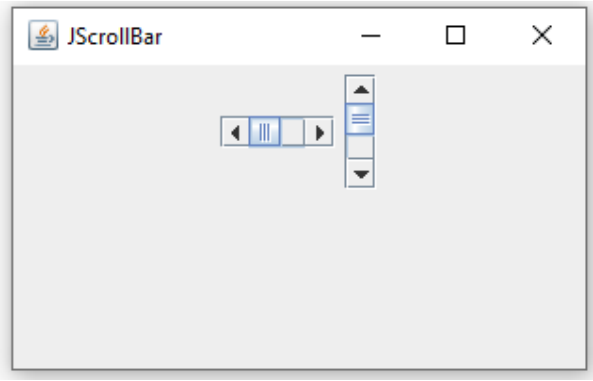


Figure 9. JScrollBar Example

Check your progress 3

1. Which of the following components can display images?

a. JTextArea

b. JTable

c. JList

d. JLabel

2. Which of the following methods is used to set the title of a JFrame?

a. setTitle()

b. setLabel()

c. setText()

d. setCaption()

3. Which of the following methods is used to create a new JMenuBar object?

a. new JMenuBar()

b. createMenuBar()

c. makeMenuBar()

d. getMenuBar()

4. Which of the following methods is used to add a menu to a JMenuBar object?

a. addMenu()

b. createMenu()

c. makeMenu()

d. add()

5. To create swing components, which of the following is imported?

a. javax.awt.*

b. java.awt.*

c. javax.swing.*

d. java.swing.*

6. Swing elements are preceded by the letter

a. S

b. A

c. X

d. J

1.6 Let Us Sum Up

Swing components are the fundamental building blocks of java GUI application. We have discussed important swing components of Java swing classes with example. Using all the components which comes with swing, it becomes easier for users to build optimized GUI applications. Buttons and labels can be displayed with images instead of or in addition to text. The borders around most Swing components can be changed easily. Swing components do not have to be rectangular. For example, Buttons can be round. Swing also supports a pluggable look and feels. Apart from the components discussed in this unit Swing provides more powerful components such as JTable, JList, JColourChooser, JTabbedPane, JFileChooser , JProgressBar, JSpinner etc.

1.7 Answer for Check Your Progress

Check your progress 1: 1. a 2. c 3. b 4. c 5. c
Check your progress 2: 1. d 2. a 3. b
Check your progress 3: 1. d 2. a 3. A 4. d 5. c 6. d

1.8 Glossary

1. Component: It is not a window. It is an abstract class underlying Buttons etc.

2. Container: These are the platforms on which all the other windows are built. They manage the child Components and LayoutManager.

3. Dialog: It is a pop-up box to deliver an error message or alert. Temporary Window for displaying information or requesting keystrokes. It needs a parent Frame, thus it cannot be used inside an Applet which has no Frame. It can be modal, which means it blocks input to all other Windows until it is dismissed. It requires having a Frame mentioned in the constructor.

4. Frame: It is a resizable, movable window with title bar and close button. Usually it contains Panels.

5. Panel: It is an area internal to a Frame or another Panel. It is used for grouping components together. It has no visible border. User can change background colour of a panel to delimit it though. It is contained inside some enclosing Container.

1.9 Assignment

1. Define GUI. Discuss various advantages of GUI Application.
2. Differentiate Swing components from AWT components.

1.10 Activities

- Explore the JTable component and its various methods.
- Explore the JTree component and its various methods.
- Explore JTabbedPane component and its various methods.

1.11 Case Study

- Try and Understand Look & Feel mechanism of Swing.

1.12 Further Readings

- https://www.codingninjas.com/codestudio/library/swing-components-in-java
- https://www.mindprod.com/jgloss/swing.html
- https://www.educba.com/swing-components-in-java/
- https://www.javatpoint.com/java-swing

UNIT 2: EVENT HANDLING AND LAYOUT

Unit Structure

2.0 Learning Objectives

After learning this Unit, you will be:

- Able to design interactive GUI application
- Able to define event handlers for various kinds of events
- Able to arrange component on containers

2.1 Introduction

Any program that uses GUI (graphical user interface) such as Java application written for windows, is event driven. Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the GUI components. For example, clicking on a button, dragging the mouse, entering a character via keyboard, selecting an item from list, scrolling the page are the activities that causes an event to occur. The events can be broadly classified into two categories. First, Foreground Events are those events that require the direct interaction of user. They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page etc. Second, Background Events are those events that don't require the interaction of end user to generate are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events. In this unit we are going to discuss various kinds of events and the mechanism to handle the events. We will also discuss various types of Layout to arrange the component on the containers.

2.2 Event Delegation Model

Earlier in the Java 1.0, the event model for event processing was based on the concept of containment. In this approach, whenever a user initiates an event, it is first sent to the component in which the event has happened. But in case the event is not handled at this component, it is automatically propagated to the container of that component. This process continued until the event is handled or it reaches the top of the container hierarchy. The major drawback in this approach is that events could be handled by the component that generated it or by the container of that component. Another major problem is that events are frequently sent to those components that cannot process them, thus wasting a lot of CPU cycles.

The advanced versions of Java ruled out the limitations of Java 1.0 event model. This model is known as the Event Delegation Model which defines a logical approach to handle events. It is based on the concept of source and listener. A source triggers an event and sends it to one or more listeners. On receiving the event, listener handles the event and returns it. The important feature of this model is that the source has a registered list of listeners which will receive the notification as they occur. Only the listeners that have been registered actually receive the notification when a specific event occurs.

For example, whenever the mouse is clicked on Button, an event will be generated. If the Button has a registered listener to handle the event, this event notification will be send to registered listener, event will be processed and the output will be returned to the user. However, if it has no registered listener the event will not be propagated upwards to the container i.e. Panel or Frame. Three important aspects of Event Delegation Model are:

Event Source:

An event source is an object that is responsible to generate a particular kind of event. An event is generated when the internal state of the event source is changed. A source may generate more than one type of event. Every source must register a list of listeners that are interested to receive the event notifications regarding the type of event. Generally, the event source is a button or the other component that the user can interact but any Swing component can be an event source. The task of the event source is to accept registrations, get events from the user and call the concerned listener's event handling method. Event source provides methods to add or remove listeners. The general form of method to register (add) a listener is:

- public void addTypeListener(TypeListener eventlistener)

Similarly, the general form of method to unregister (remove) a listener is:

- public void removeTypeListener(TypeListener eventlistener)

where,

Type is the name of the event and eventlistener is a reference to the event listener.

Event Listener:

An event listener is an object which receives notification when an event occurs. As already said, only registered listeners will receive event notification from sources about specific type of events. The main task of an event listener is to implement the interface, register with the source and provide the event handling mechanism.

Event Handler:

An event handler is a function or method that executes program logic in response to an event. A software program that processes activities such as keystrokes or mouse movements is what an event handler is.

Check your progress 1

1. To manage events, Java employs the ____?

a. Custom-based Event Model

b. Retired Event Model

c. Delegation Event Model

2. How many types of events are there?

a. 5

b. 3

c. 2

d. 4

3. Which of the following type of events requires direct interaction with the user?

a. Foreground events

b. Background events

2.3 Types of Events and Event Handlers

As we discussed in Event Delegation Model, source is responsible for generating an event. There are various sources who generates different kinds of events. For such events there are various kinds of corresponding event handlers to handles those events. The below table lists the various event classes within the java.awt.event package, their listener interface, methods and the components associated with each of them.

No	Event Class	Listener Interface	Event Handlers / Methods	Descriptions
1	ActionEvent	ActionListener	actionPerformed()	When a button is clicked or a list item is double-clicked, an ActionEvent is triggered.
2	AdjustmentEvent	AdjustmentListener	adjustmentValueChanged()	when the scroll bar is manipulated
3	ComponentEvent	ComponentListener	componentResized(), componentMoved(), componentShown() and	when a component is hidden, moved, resized, or made visible

			componentHidden()	
4	ContainerEvent	ContainerListener	componentRemoved() and componentAdded()	when a component is added or removed from a container
5	FocusEvent	FocusListener	focusLost() and focusGained()	when a component gains or losses keyboard focus
6	ItemEvent	ItemListener	itemStateChanged()	An event that indicates whether an item was selected or not.
7.	KeyEvent	KeyListener	keyPressed(), keyReleased(), and keyTyped().	The Key event is triggered when the character is entered using the keyboard.
8	MouseEvent	MouseListener and MouseMotionListener	mouseClicked(), mousePressed(), mouseEntered(), mouseExited() and mouseReleased() are the mouseListener methods. mouseDregged() and mouseMoved() are the MouseMotionListener methods.	This event indicates a mouse action occurred in a component
9.	MouseWheelEvent	MouseWheelListener	mouseWheelMoved()	generated when the mouse wheel is rotated
10	TextEvent	TextListener	textChanged()	When the value of a textarea or text field is changed
11.	WindowEvent	WindowListener	windowActivated(), windowDeactivated(), windowOpened(), windowClosed(), windowClosing(), windowIconfied() and windowDeiconified().	The object of this class represents the change in the state of a window and are generated when the window is activated, deactivated, deiconified, iconified, opened or closed

2.4 Working with Event Handling

The following steps are required to perform event handling in Java:

- Implement appropriate interface in the class:

The first step is to implement an appropriate interface in the class.

- Register the component with the listener:

Once the implementation of the interface in the class is done, the second step is to register the created components with listeners, which can be done using inbuilt functions.

Now, we will see different event handling mechanism with the help of examples.

The ActionListener Interface

The ActionListener listener interface is used for receiving action events. Any class interested in processing action events should implement this interface. An object created with the class is registered with a component using that component's addActionListener() method. When the action event occurs on the component we have registered, the actionPerformed(ActionEvent e) method of the object is invoked.

Example: **SwingComponent.java**

In the following example two buttons and one Label component are created. Label component text changes each time the button is clicked.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SwingComponent extends JFrame implements ActionListener
{
        JLabel status;
        JButton yes,no;
  public SwingComponent()
  {
          Container cp = getContentPane();
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     yes = new JButton("Yes");
     no = new JButton("No");
    status=new JLabel();
     cp.setLayout(new FlowLayout());
     cp.add(yes);
     cp.add(no);
     cp.add(status);
     yes.addActionListener(this);   // Listener Registration
     no.addActionListener(this);    // Listener Registration
```

```
 }
 public void actionPerformed(ActionEvent ae)    // Event Handler
 {
            String command = ae.getActionCommand();
      // code to check which button is pressed
      if( command.equals( "Yes" ))
      {
         status.setText("Yes Button clicked.");
      }
      else
      {
        status.setText("No Button clicked.");
      }
 }
 public static void main(String[] args)
 {
      JFrame jframe = new SwingComponent ();
      jframe.setTitle("ActionEvent Example");
      jframe.setSize(300,200);
      jframe.setVisible(true);
  }
}
```

In the above program we have created a frame and a button and registered the button with the invoking object (this), which is our implementation class i.e SwingComponent. We then implement the only method within the ActionListener interface which is actionPerformed(ActionEvent e). Every time we press the button this method is invoked and its message is displayed in the label's text, so we can see it change each time. Similarly we can implement different listeners for various components and its corresponding events.

After successful compilation, it will have following output when we execute above program:

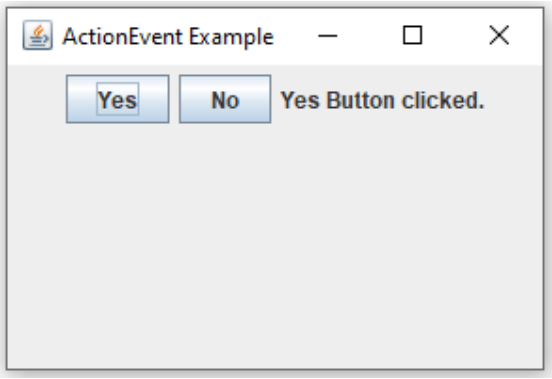


Figure 1. ActionEvent Example

The AdjustmentListener Interface

The AdjustmentListener listener interface is used for receiving adjustment events. Any class interested in processing adjustment events should implement this interface. A component created with the class is registered using component's addAdjustmentListener() method. When the adjustment event occurs on the component we have registered, the adjustmentValueChanged(AdjustmentEvent e) method of the object is invoked.

Example: **SwingAdjustmentListener.java**

In the following example we have created a text area on the left of the frame to receive messages when the scrollbar on the right of the Jframe is interacted with.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SwingAdjustmentListener extends JFrame implements AdjustmentListener
{
   JTextArea ta;
   public SwingAdjustmentListener()
  {
    Container cp = getContentPane();
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     ta = new JTextArea("Welcome to BAOU JTextArea.");
    cp.setLayout(new FlowLayout());
      ta.setEditable(false);
      ta.setLineWrap(true);
      ta.setWrapStyleWord(true);
```

```
    // Create and register a JScrollbar
    JScrollBar vbar = new JScrollBar(JScrollBar.VERTICAL, 0, 50, 0, 220);
       cp.add(ta);
       cp.add(vbar);
       vbar.addAdjustmentListener(this);
  }
  // Implement the method of AdjustmentListener Interface
  public void adjustmentValueChanged(AdjustmentEvent e)
       {
              ta.append("This text is Appended. ");
       }
       public static void main (String[] args)
       {
    JFrame jframe = new SwingAdjustmentListener();
    jframe.setTitle("AdjustmentEvent Example");
    jframe.setSize(350,250);
    jframe.setVisible(true);
  }
}
```

The following figure 2 shows the results of compiling and running the OurAdjustmentListener class and scrolling the right scrollbar a few times. We implements the AdjustmentListener on SwingAdjustmentListener class. We then creates a scrollbar and text area. We then add a scroll bar to container and register it with the invoking object (this), which is our SwingAdjustmentListener. After this we implements the only method of AdjustmentListener interface which is adjustmentValueChanged(AdjustmentEvent ae). Every time we press or move the scroll bar this method is invoked and it append some text to the text area on the left. After successful compilation, it will have following output when we execute above program:

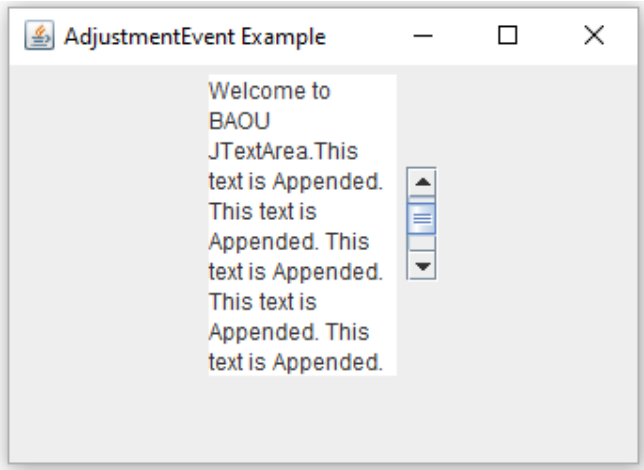


Figure 2. AdjustmentEvent Example

The ItemListener Interface

The ItemListener listener interface is used for receiving item selection events. Any class interested in processing item selection events should implement this interface. An object created with the class is registered with a component using that component's addItemListener() method. When an item selection event occurs on the component we have registered, the itemStateChanged(ItemEvent e) method of the object is invoked.

Example: **SwingItemListener.java**

In the following example we create a text area on the left of the frame to receive messages when the radio buttons are interacted with.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SwingItemListener extends JFrame implements ItemListener
{
   JRadioButton jrb1,jrb2,jrb3,jrb4;
   JTextArea ta;
   public SwingItemListener()
         {
     Container cp = getContentPane();
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          ta = new JTextArea("Welcome to BAOU JTextArea.",1,30);
```

```
        cp.setLayout(new FlowLayout());
    ta.setLineWrap(true);
    ta.setWrapStyleWord(true);
    // Create some radio buttons
    jrb1 = new JRadioButton("Ahmedabad");
    jrb2 = new JRadioButton("Vallabh Vidyanagar");
    jrb3 = new JRadioButton("Vadodara");
    jrb4 = new JRadioButton("Gandhinagar");
    // Group the buttons so only one can be selected at a time
    ButtonGroup bg = new ButtonGroup();
    bg.add(jrb1);
       bg.add(jrb2);
       bg.add(jrb3);
       bg.add(jrb4);

   JPanel panel = new JPanel();
    panel.add(jrb1);
    panel.add(jrb2);
    panel.add(jrb3);
    panel.add(jrb4);
   cp.add(ta);
   cp.add(panel);
   // Register the radio buttons with our ItemListener object
    jrb1.addItemListener(this);
    jrb2.addItemListener(this);
    jrb3.addItemListener(this);
    jrb4.addItemListener(this);

}
public void itemStateChanged(ItemEvent e)
      {
    Object o = e.getSource();
    if (o instanceof JRadioButton)
```

```
                {
        // Downcast Object to JRadioButton so we can use methods of JRadioButton
class
        ta.append("You selected: " + ((JRadioButton)o).getText() + "\n");
      }
   }
        public static void main (String[] args)
        {
      JFrame jframe = new SwingItemListener();
      jframe.setTitle("ItemEvent Example");
      jframe.setSize(500,250);
      jframe.setVisible(true);
   }
}
```

The following figure 3 shows the results of compiling and running the SwingItemListener class. We implement the ItemListener on our SwingItemListener class. We then created a text area and some radio buttons which we put in a button group so that only one can be selected at a time. We also grouped all the radio buttons in one panel. We registered all the radio buttons with the invoking object (this), which is our SwingItemListener. After this we implemented the method of the ItemListener interface which is itemStateChanged(ItemEvent e). Every time we select a radio button this method is invoked and we append some text to the text area. The text appended gets duplicated because as we change radio buttons two events are triggered, one for the radio button being deselected and another for the radio button being selected. After successful compilation, it will have following output when we execute above program:

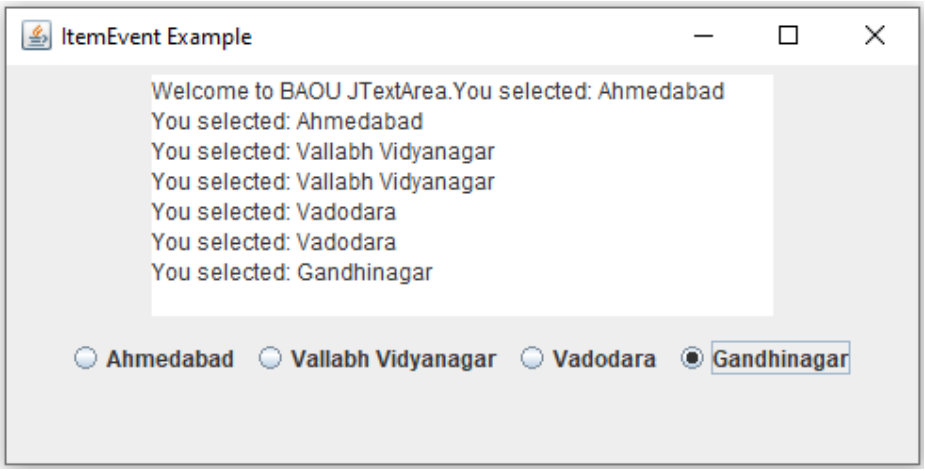


Figure 3. ItemEvent Example

The MouseListener Interface

Mouse event happens when a mouse related activity is performed on a component such as clicking, dragging, pressing, moving, dragging or releasing a mouse etc. Objects representing mouse events are created from MouseEvent class. There are two listener interfaces corresponding to the MouseEvent Class. These are MouseListener and MouseMotionListener interface.

Example: **SwingMouseComponent.java**

In the following example we created a text area and implemented various mouse related methods.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SwingMouseComponent extends JFrame implements MouseListener
{
   JLabel lblmsg;
   JTextArea txtvalues;
   SwingMouseComponent()
   {
       Container cp = getContentPane();
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
           cp.setLayout(new FlowLayout());
                   JPanel panel1 = new JPanel(new FlowLayout());
```

```
        JPanel panel2 = new JPanel(new FlowLayout());
        lblmsg = new JLabel("Press,Release or Click the Mouse on the txtarea to display
x,y Coordinates");
        txtvalues = new JTextArea(10,30);
        panel1.add(lblmsg);
        panel2.add(txtvalues);
        cp.add(panel1);
        cp.add(panel2);
                    txtvalues.addMouseListener(this);
    }
    public void mousePressed(MouseEvent e)
    {
        String s= "x-Corrdinate = " + e.getX() + "y-Coordinate = " + e.getY();
        System.out.println("Mouse Pressed");
        txtvalues.setText(s);
    }

    public void mouseReleased(MouseEvent e)
    {
        String s = "x-Coordinate = " + e.getX() + "y-Coordinate = " + e.getY();
        System.out.println("Mouse Released");
        txtvalues.setText(s);
    }
    public void mouseClicked(MouseEvent e)
    {
        String s= "X-Corrdinate = " + e.getX() + " y-Coordinate = " + e.getY();
        System.out.println("Mouse Clicked");
        txtvalues.setText(s);
      }
        public void mouseEntered(MouseEvent e)
          {
            System.out.println("Mouse Entered");
          }
```

```
        public void mouseExited(MouseEvent e)
        {
           System.out.println("Mouse Exited");
        }
 public static void main(String[] args)
 {
     JFrame jframe = new SwingMouseComponent ();
     jframe.setTitle("MouseEvent Example");
     jframe.setSize(450,200);
     jframe.setVisible(true);
  }
}
```

The following figure 4 shows the results of compiling and running the SwingMouseComponent class. We have implemented the MouseListener on our SwingMouseComponent class. Whenever a mouse is Pressed and clicked on the text area, its x & y co-ordinate values will be displayed in the text area along with event messages on the command prompt. After successful compilation, it will have following output when we execute above program:

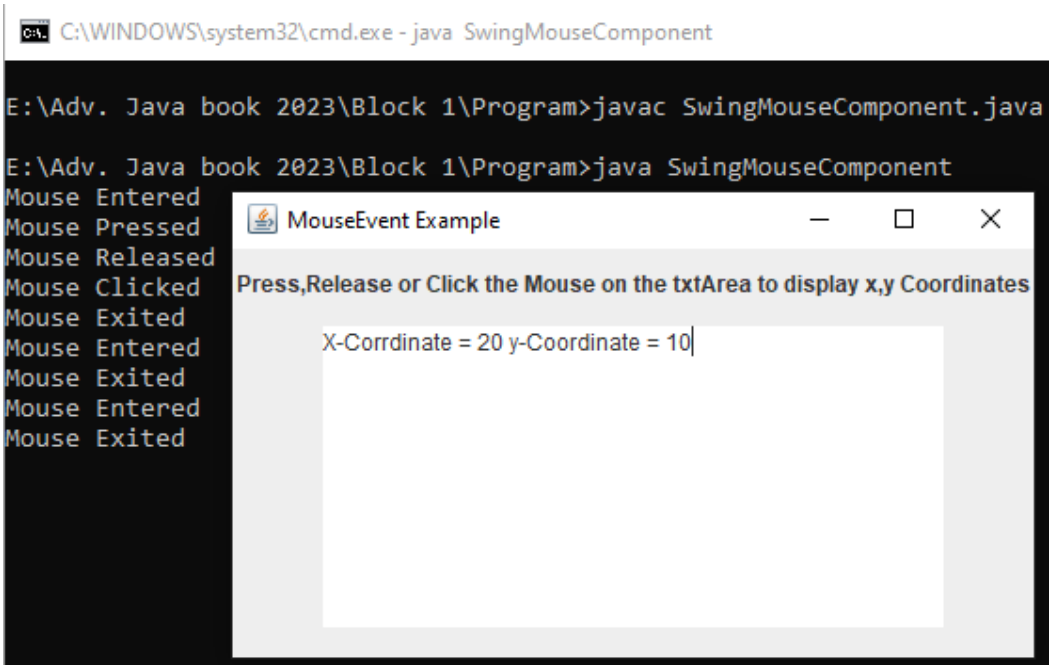


Figure 4. MouseEvent Example

Similarly we can implement MouseMotionListener interface if mouse is moved or dragged.

Check your progress 2

1. Which of the following events is fired when a component gains focus?

a. mouseClicked

b. mouseEntered

c. focusGained

d. focusLost

2. Which of the following events is fired when a button is clicked?

a. mousePressed

b. mouseReleased

c. actionPerformed

d. keyPressed

3. Which of the following methods is used to add an ActionListener to a JButton object?

a. addAction()

b. addActionListener()

c. addListener()

d. addEventHandler()

4. An ____ is a change in the state of an item?

a. Spinner

b. Event

c. Occurrence

d. Activity

5. Which of the following events is generated when the size, position or visibility of a component is changed?

a. Key Event

b. Component Event

c. Container Event

d. Focus Event

6. Which event is generated when a checkbox or a list item is clicked or when a check menu item is selected or deselected?

a. Container Event

b. Mouse Event

c. Focus Event

d. Item Event

2.5 Layout Manager

Layout Managers is used for arranging the components on the container in order. LayoutManager is an interface that is implemented by all the classes of layout managers. It is the class that is responsible for determining the size and position of each component within a container based on a set of rules. The layout manager in java takes into consideration the size of the container and the preferred size of the components.

Types of the Layout Manager

Java provides various built-in layout managers that can be used to arrange components within a container. Important Layout Mangers are discussed below:

Flow Layout:

A FlowLayout arranges components in a row, adding additional rows as needed when the width of the container is exceeded. From left to right, the components are added, with the next component being added directly to the right of the one before it. There are 3 types of constructor in the Flow Layout. They are as following:

1. FlowLayout()
2. FlowLayout(int align)
3. FlowLayout(int align, int hgap, int vgap)

Example: **flowLayout.java**

In the following example we have demonstrated the use of FlowLayout Manager.

```
import javax.swing.*;
import java.awt.*;
public class flowLayout extends JFrame
{
   public flowLayout()
         {
                  Container cp = getContentPane();
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     cp.setLayout(new FlowLayout());
           JPanel panel = new JPanel();
                  panel.add(new JButton("Java"));
```

```
        panel.add(new JButton("DBMS"));
        panel.add(new JButton("C++"));
        panel.add(new JButton("DotNet"));
        panel.add(new JButton("Python"));
                cp.add(panel);
           }
           public static void main(String[] args)
           {
        JFrame jframe = new flowLayout();
                jframe.setTitle("FlowLayout Example");
        jframe.setSize(500,200);
        jframe.setVisible(true);
       }
}
```

The following figure 5 shows the results of compiling and running the flow layout. We have added five buttons in the class and they were positioned using flow layout. Note that by default, the FlowLayout layout manager centers components horizontally in the container. To change this behaviour, you can use the setAlignment() method on the layout manager. After successful compilation, it will have following output when we execute above program:

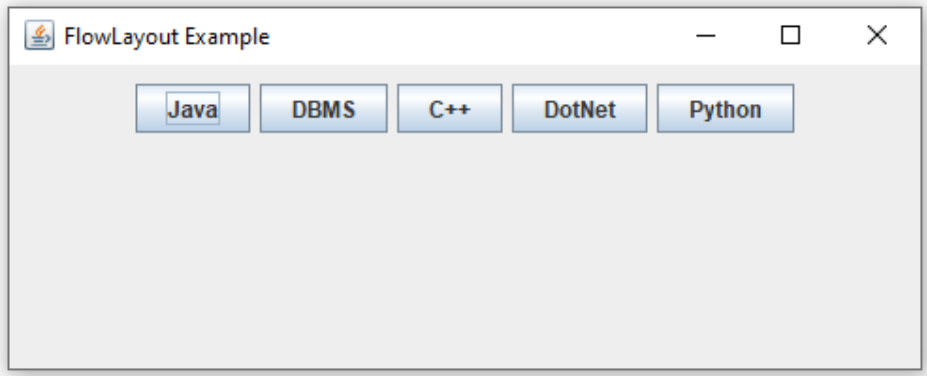


Figure 5. FlowLayout Example

Border Layout:

The BorderLayout divides the container's five regions into the NORTH, SOUTH, EAST, WEST and CENTER. Whenever a component is added to the container, it is put in one of these regions and fills the entire region. The old component is replaced by the new one if a component is added to a region that already has one.

Example: **borderLayout.java**

In the following example we have demonstrated the use of BorderLayout Manager.

```
import javax.swing.*;
import java.awt.*;
public class borderLayout extends JFrame
{
   public borderLayout()
         {
                  Container cp = getContentPane();
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JPanel panel = new JPanel();
            panel.setLayout(new BorderLayout());
                  panel.add(new JButton("Java"),BorderLayout.NORTH);
      panel.add(new JButton("DBMS"),BorderLayout.SOUTH);
      panel.add(new JButton("C++"),BorderLayout.EAST);
      panel.add(new JButton("DotNet"),BorderLayout.WEST);
      panel.add(new JButton("Python"),BorderLayout.CENTER);
                  cp.add(panel);
         }
         public static void main(String[] args)
         {
      JFrame jframe = new borderLayout();
                  jframe.setTitle("BorderLayout Example");
      jframe.setSize(500,200);
      jframe.setVisible(true);
     }
}
```

The following figure 6 shows the results of compiling and running the border layout. We have created a JPanel and set the layout manager of the panel to BorderLayout. We then created and added five buttons to the panel, each in a different region of the container (north, south, west, east and center). Note that the Center component takes up all the remaining space in the container after the other components have been placed. After successful compilation, it will have following output when we execute above program:

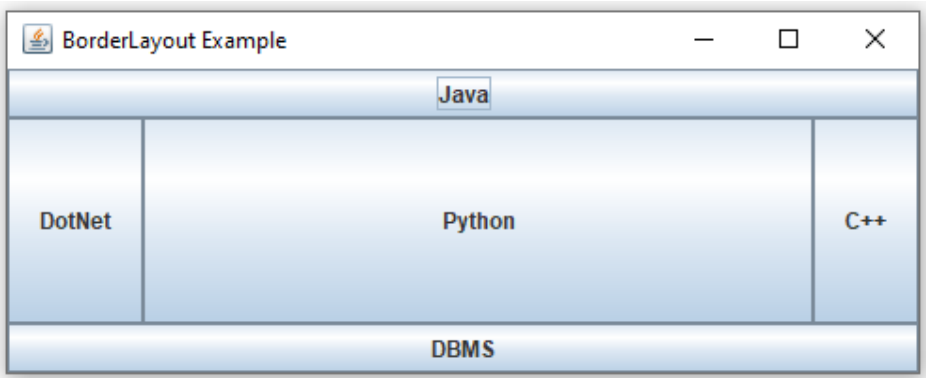


Figure 6. BorderLayout Example

Grid Layout:

The GridLayout arranges elements in a grid of rows and columns. The layout manager is created with a specified number of rows and columns, and components are added one at a time, filling each grid cell from left to right and from top to bottom. There are 3 types of constructor in Grid Layout. They are as following:

1. GridLayout()
2. GridLayout(int rows, int columns)
3. GridLayout(int rows, int columns, inthgap, int vgap)

Example: **gridLayout.java**

In the following example we have demonstrated the use of GridLayout Manager.

```
import javax.swing.*;
import java.awt.*;
public class gridLayout extends JFrame
{
   public gridLayout()
  {
      Container cp = getContentPane();
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JPanel panel = new JPanel();
      panel.setLayout(new GridLayout(3,3));
     panel.add(new JButton("Java"));
      panel.add(new JButton("DBMS"));
      panel.add(new JButton("C++"));
      panel.add(new JButton("DotNet"));
      panel.add(new JButton("Python"));
```

```
        panel.add(new JButton("Cloud"));
        panel.add(new JButton("ML"));
        panel.add(new JButton("Data Science"));
        panel.add(new JButton("SQL"));
        cp.add(panel);
    }
        public static void main(String[] args)
        {
                JFrame jframe = new gridLayout();
                jframe.setTitle("GridLayout Example");
                jframe.setSize(500,200);
              jframe.setVisible(true);
     }
}
```

The following figure 7 shows the results of compiling and running the grid layout. We have created a JPanel and set the layout manager of the panel to GridLayout. We have created a JPanel and set the layout manager of the panel to GridLayout with three rows and three columns. We then created and added nine buttons to the panel. When we run the program, we can see a window with the buttons arranged in a 3×3 grid. After successful compilation, it will have following output when we execute above program:

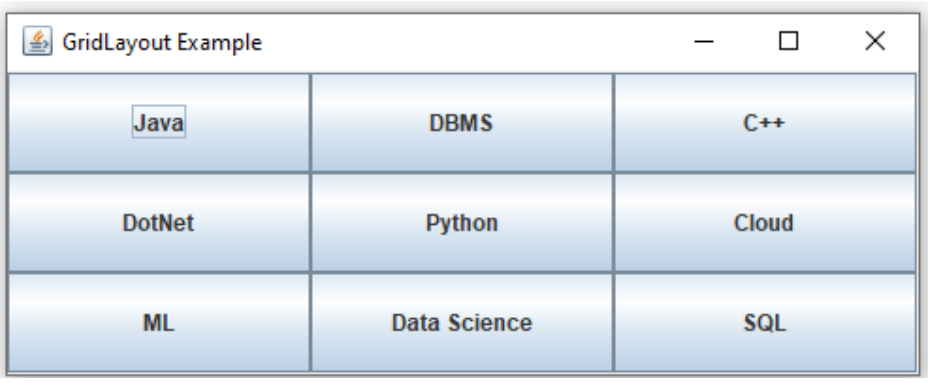


Figure 7. GridLayout Example

Card Layout:

CardLayout is a layout manager just like a tabbed pane that enables switching between multiple components while keeping them in the same container by using functions like next() and previous (). A single component can be seen at a time, and the CardLayout can be used to

switch between various views of the same data or to create an interface that looks like a wizard. There are 2 types of constructor in the Card Layout. They are as following:

1. CardLayout()
2. CardLayout(inthgap, intvgap)

Example: **cardLayout.java**

In the following example we have demonstrated the use of CardLayout Manager.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class cardLayout extends JFrame implements ActionListener
{
        CardLayout CL;
        JButton card1,card2,card3;
        Container cp;
        cardLayout()
        {
                cp=getContentPane();
                CL=new CardLayout(20,20);
                cp.setLayout(CL);
                 card1=new JButton("Java");
                 card2=new JButton("Cloud");
                card3=new JButton("DBMS");
                card1.addActionListener(this);
                card2.addActionListener(this);
                card3.addActionListener(this);
                cp.add("First",card1);
                cp.add("Second",card2);
                cp.add("Third",card3);
  }
  public void actionPerformed(ActionEvent e)
        {
                CL.next(cp);
        }
```

```
  public static void main(String[] args)
  {
                JFrame obj =new cardLayout();
                obj.setSize(300,200);
                obj.setTitle("CardLayout Example");
                obj.setVisible(true);
                obj.setDefaultCloseOperation(EXIT_ON_CLOSE);
        }
}
```

The following figure 8 shows the results of compiling and running the card layout. We have created a JPanel and set the layout manager of the panel to GridLayout. We have created a JPanel and set the layout manager of the panel to GridLayout with three rows and three columns. We then created and added nine buttons to the panel. When we run the program, we can see a window with the buttons arranged in a 3×3 grid. After successful compilation, it will have following output when we execute above program:

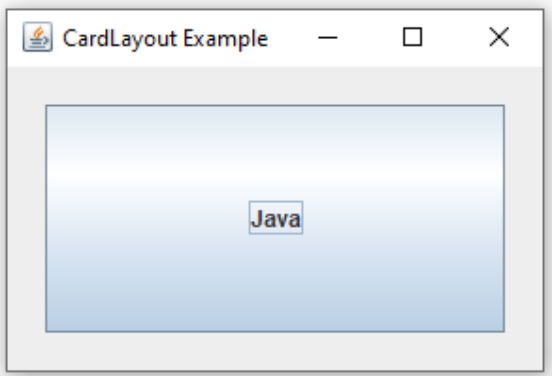


Figure 8. CardLayout Example

When we click on Java button it will show next card as shown in figure 9. Likewise we can swap to next and previous cards.

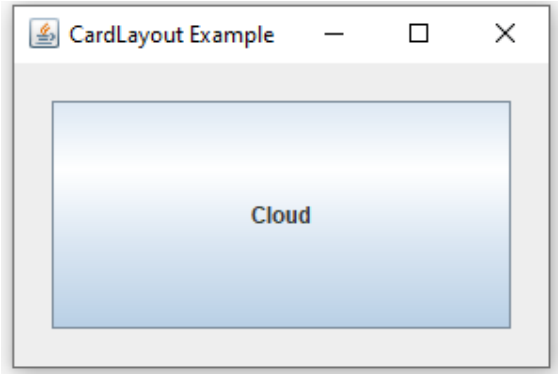


Figure 9. CardLayout Example

GridBag Layout:

A GridBagLayout arranges elements in a versatile grid of rows and columns. As opposed to GridLayout, components can span multiple rows or columns and be positioned in specific places within the grid. Because of this, GridBagLayout is not only more capable and flexible than other layout managers but also trickier to use.

Group Layout:

GroupLayout layout manager enables programmers to specify a container's layout by building a nested hierarchy of groups. To regulate the size and placement of the components within the container, the developer can specify constraints for each group, which each group can contain other groups or components. Although GroupLayout is a more capable and adaptable layout manager than other layout managers, it is also trickier to use.

Check your progress 3

1. Which layout manager arranges components in a grid?

a. BorderLayout

b. FlowLayout

c. GridLayout

d. CardLayout

2. What is the purpose of the GroupLayout layout manager?

a. To arrange components in a grid

b. To arrange components in a circular layout

c. To arrange components in a nested layout

d. To arrange components based on their preferred sizes

2.6 Let Us Sum Up

The benefit of event handling mechanism is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to the separate piece of code. In this mechanism, Listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to those listeners that are interested to receive them. Event handling in Java is controlling an event and taking appropriate action if one occurs. Events handling makes web pages and mobile applications live. LayoutManager makes the task easy for application GUI designer in arranging the components on the containers.

2.7 Answer for Check Your Progress

Check your progress 1: 1. c 2. c 3. a
Check your progress 2: 1. c 2. c 3. b 4. b 5. b 6. d
Check your progress 3: 1. c 2. d

2.8 Glossary

1. Listener Interface: It is an interface which contains methods that the listener must implement and the source of the event invokes when the event occurs.

2. Event Handling: Event Handling is the mechanism that controls the event and decides what should happen if an event occurs.

3. Layout Manager: The Layout Managers are used to arrange components in a particular manner.

2.9 Assignment

1. Define Event. Explain Event handling mechanism of java.
2. Discuss the various Listener interfaces.
3. Discuss the importance of different layout managers.

2.10 Activities

- Explore the mechanism of key event handling using KeyListener.

- Explore the mechanism of window event handling using WindowListener.

2.11 Case Study

- Try and understand the different situation of layout manager applicability.

2.12 Further Readings

- https://www.tutorialspoint.com/swing/swing_event_handling.htm
- https://www.scaler.com/topics/event-handling-in-java/
- https://server2client.com/javaswing/eventhandling.html
- https://www.javatpoint.com/java-layout-manager

UNIT 3: JAVA NETWORKING

Unit Structure

3.0 Learning Objectives

After learning this Unit, you will be able to:

- Define various Networking Terminology
- Define various Networking Protocols
- Differentiate among TCP and UDP
- Write TCP server and client program

3.1 Introduction

The concept network programming is associated with writing programs that will execute over various computer devices connected with each other to share resources using a network. One important aspects of java is that it incorporates an easy-to-use, cross-platform model for network communications. Java Networking allows us to connect two or more computing devices together to share resources. Java program communicates over the network at the application layer. Java.net package contains all the useful Java networking related classes and interfaces. In this unit we will discuss various networking related terminology, protocols and client-server programming using socket. The data that is sent back and forth over a socket can be anything.

3.2 Java Networking Terminology

The widely used java networking terminologies are given below:

- **IP Address:**

The IP address is a unique number assigned to a system of a network e.g. 192.168.0.5. It distinguishes a device on the internet or a local network. It is composed of octets and can be changed. It is referred to as a logical address. For example, each time you access the network, your device will be assigned a new IP address by the server through DHCP (dynamic host configuration protocol). This address will be used to route your data through the network from the source device to the desired destination. It exists at the network layer. IP address is divided in classes from A to E. The IP address 127.0.0.1 is special, and is reserved to represent the loopback or localhost address. The range of the IP Address is from 0.0.0.0 to 255.255.255.255.

- **Protocol:**

Protocol defines the rules and conventions for communication between network devices, including ways devices can identify and make connections with each other. They are the

reason through which a user can easily communicate with other users across the world and thus play a critical role in modern digital communications. Examples of protocol are TCP, FTP, Telnet, SMTP, POP etc.

- **Port Number:**

The port number helps in uniquely identifying different applications. It behaves as a communication endpoint between applications. Along with an IP Address the port number is used to communicate between two applications. Port works at the transport layer. Port numbers 1 to 255 are reserved by IP for well-known services. If user tries to connect to port 80 of a host, for example, may expect to find an HTTP server. On unix machines, ports less than 1024 are privileged and can only be bound by the root user.

- **MAC Address:**

The MAC address is the physical address of the system (e.g. network interface card). It is fixed and each device in the world has a unique MAC address. MAC addresses works at the data-link layer. It contains a 48 bit or 64-bit address, combined with the network adapter. It can be in hexadecimal composition. Simply, a MAC address is a unique number that is used to track a device in a network.

- **Connection-Oriented and Connection-Less Protocol:**

In the connection-oriented protocol, user must establish a connection before starting the communication. Once data is send, the acknowledgment is sent by the receiver. So it is reliable but slow. The example of a connection-oriented protocol is TCP. But, in the connection-less protocol, the data is communicated in one route from source to destination without verifying that the destination is there or not. Authentication is not needed in the connectionless protocol. Here, the acknowledgment is not sent by the receiver. So it is not reliable but fast. The example of a connection-less protocol is UDP.

- **Socket:**

A socket in Java is one endpoint of a two-way communication link between two programs running on the network. Socket classes are used to create a connection between a client program and a server program. A socket is tied to a port number so that the TCP layer can identify the application where the data is supposed to be sent.

3.3 Common Network Protocols

Networking protocols are sets of established rules that describe how to format, transmit and receive data from servers and routers to endpoints so computer network devices can

communicate regardless of the differences in their underlying infrastructures, designs or standards.

Devices on both sides of a communication must accept and follow protocol rules to successfully send and receive information. The java.net package provides the networking support. All the classes for making a network program are defined in the java.net package. Through TCP we can communicate over the network. Typically a client opens a TCP/IP connection to a server. The client then starts to communicate with the server. When the client finishes its task, it closes the connection again.

If there are more round trips in your protocol, it makes the protocol slower as the latency is high. The HTTP protocol consists of only a single request and a single response to perform its service involving a single roundtrip. The SMTP protocol on the other hand, consists of several roundtrips between the client and the server before an email is sent. There are other protocols used in network programming like Telnet, FTP, POP, HTTPS etc. The java.net package provides the functionality for two common protocols.

TCP (Transmission Control Protocol)

TCP is a connection based protocol that provides a reliable flow of data between two devices. It allows secure communication between different applications. TCP is a connection-oriented protocol which means that once a connection is established, data can be transmitted in two directions. It is typically used over the Internet Protocol. This protocol provides the reliable connections between two applications to communicate easily.

UDP (User Datagram Protocol)

UDP protocol sends independent packets of data, called datagram from one computer to another with no guarantee of arrival. It is connection less protocol. It is a simply Internet protocol in which error-checking and recovery services are not required. In UDP, the data is continuously sent to the recipient whether they receive it or not.

Difference between TCP & UDP:

The important differences between TCP and UDP are discussed in table 1:

Parameter	TCP	UDP
Service type	It is a connection-oriented protocol.	It is a connection-less protocol.
Reliability	TCP is reliable as it guarantees the delivery of data to the destination.	The delivery of data to the destination cannot be

Parameter	TCP	UDP
		guaranteed.
Error checking mechanism	TCP provides extensive error-checking mechanisms.	UDP has only the basic error-checking mechanism using checksums.
Acknowledgment	An acknowledgment segment is present.	No acknowledgment segment.
Sequence	Sequencing of data is a feature of Transmission Control Protocol (TCP). It means that packets arrive in order at the receiver.	There is no sequencing of data in UDP. If the order is required, it has to be managed by the application layer.
Speed	TCP is comparatively slower than UDP.	UDP is faster, simpler, and more efficient than TCP.
Retransmission	Retransmission of lost packets is possible in TCP, but not in UDP.	There is no retransmission of lost packets in the User Datagram Protocol (UDP).
Broadcasting	TCP doesn't support Broadcasting.	UDP supports Broadcasting.
Protocols	TCP is used by HTTP, HTTPs, FTP, SMTP and Telnet.	UDP is used by DNS, DHCP, TFTP, SNMP, RIP and VoIP.
Applications	email, web surfing and military services.	VoIP, game streaming, video and music streaming, etc.

Table 1: TCP v/s UDP

Check your progress 1

1. Which of these package contains classes and interfaces for networking?

a. java.io

b. java.util

c. java.net

d. javax.swing

2. Which of the following protocol follows connection less service?

a. UDP

b. TCP/IP

c. TCP

d. HTTP

3. Which of the following protocols is/are used for splitting and sending packets to an address across a network?

a.TCP/IP

b. FTP

c. SMTP

d. UDP

4. TCP, FTP, Telnet, SMTP, POP etc. are examples of?

a. Socket

b. IP Address

c. Protocol

d. MAC Address

3.4 Socket Programming

Sockets implement the communication tool between two devices using TCP. Java Socket programming can either be connection-oriented or connection-less. In Socket Programming, Socket and ServerSocket classes are used for connection-oriented socket programming. However, DatagramSocket and DatagramPacket classes are used for connection-less socket programming. A client program creates a socket on its end of the communication and connects that socket with a server. When the connection is established, the server creates an object of socket class on its communication end. Now, the client and the server can communicate by sending to and receiving from the socket.

The java.net.Socket class describes a socket, and the java.net.ServerSocket class implements a tool for the server program. Following are the steps to establish a TCP connection between two computing devices using Socket Programming

1. The server creates a ServerSocket object, showing the port number on which communication will takes place.

2. After creating the ServerSocket object, the server requests the accept() method of the ServerSocket class. This program will pause until a client connects to the server on the given port.

3. Once the server goes to the idle state, a client instantiates an object of Socket class, defining the server name and the port number to connect to.

4. The constructor of the Socket class attempts to connect the client to the designated server and the port number.

5. On the server-side, the accept() method returns a reference to a new socket on the server connected to the client's socket.

After the connections are estabilized, communication can happen using I/O streams. Every object of a socket class has both an OutputStream and an InputStream. The client's OutputStream is combined to the server's InputStream, and the client's InputStream is combined with the server's OutputStream. Transmission Control Protocol (TCP) is a two-way communication protocol. Hence information can be transmitted over both streams at the corresponding time.

Socket Class

This class is used to create socket objects that help the users in implementing all fundamental socket operations such as sending, reading data and closing connections. Each Socket object created using java.net.Socket class has been connected specifically with 1 remote host. If a user wants to connect to another host, then a new socket object must be created.

ServerSocket Class

This class is used for providing system-independent implementation of the server-side of a client/server Socket Connection. The constructor for ServerSocket class throws an exception if it can't listen on the specified port. For example, it will throw an exception if the port is already being used.

3.5 Client-Server Communication

Creating Server:

To create the server application, we need to create the instance of ServerSocket class. Here, we are using 1234 port number for the communication between the client and server. You may also choose any other port number. The accept() method waits for the client's request. If client connects with the given port number, it returns an instance of Socket.

```
ServerSocket ss=new ServerSocket(2222);
Socket s=ss.accept(); //establishes connection and waits for the client
```

Creating Client:

To create the client application, we need to create the instance of Socket class. Here, we need to pass the IP address or hostname of the Server and a port number to connect. Here, we are using “localhost” as our server is running on same system.

Socket s=new Socket(“localhost”, 2222);

Let's see a simple Java socket programming where client sends a text message and server reads it and then prints it.

Server Program: SockServerApp.java

```
import java.io.*;
import java.net.*;
public class SockServerApp {
public static void main(String[] args)
{
        Socket s = null;
        ServerSocket ss = null;
         DataInputStream dis = null;
        try
        {
        ss=new ServerSocket(2222);
        s=ss.accept();                          //establishes connection
        dis=new DataInputStream(s.getInputStream());
        String  str=(String)dis.readUTF();
        System.out.println("Data Read is= "+str);
        ss.close();
        }catch(Exception e){System.out.println(e);}
        }
}
```

Client Program: SockClientApp.java

```
import java.io.*;
import java.net.*;
public class SockClientApp
{
public static void main(String[] args)
{
        Socket s = null;
   DataOutputStream dout = null;
try{
s=new Socket("localhost",2222);
dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Welcome to BAOU @ Ahmedabad-Gujarat");
dout.flush();
dout.close();
```

```
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

After successfully compiling both the programs, first run the server program. It waits until client request comes. As soon as client sends requests, server will listen, read and print the data received from client. Following figure shows it in sequence.

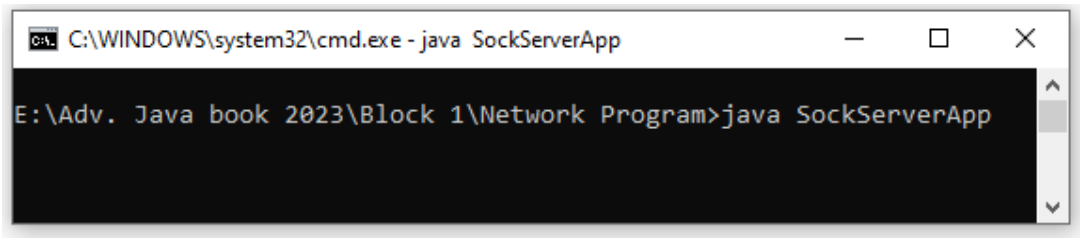


Figure 1. Running Server program

Now, run the client program from another prompt as shown in figure 2:

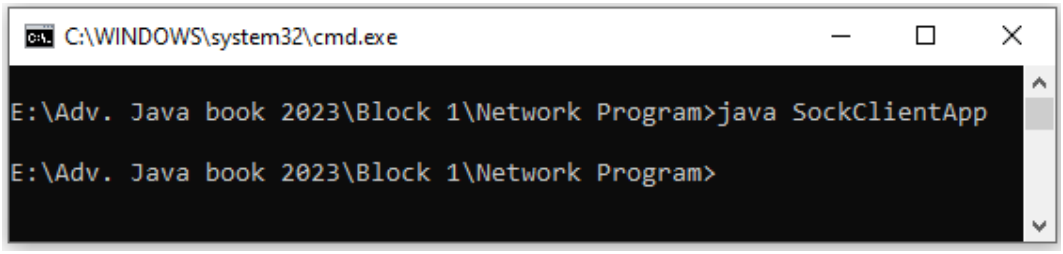


Figure 2. Running Client program

As soon as the client sends the requests, server receives it and displays the message as shown in figure 3.

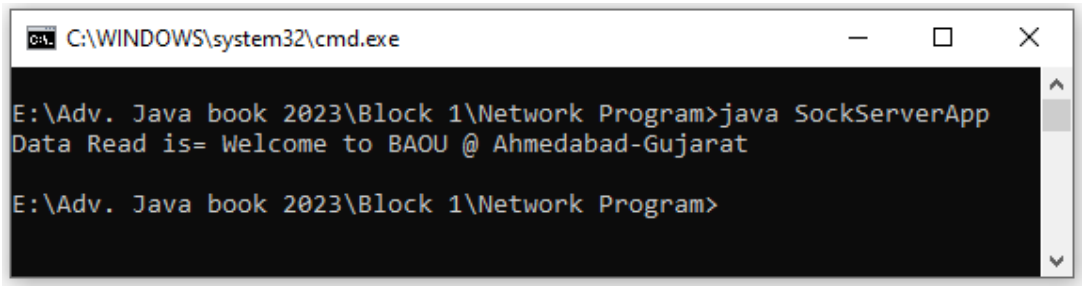


Figure 3. Server program with clients message

Check your progress 2

1. The server listens for a connection request from a client using which of the following statement?

a. Socket st = new Socket(ServerName, port);

b. Socket st = serverSocket.accept();

c. Socket st = serverSocket.getSocket();

d. Socket st = new Socket(ServerName);

2. The client requests a connection to a server using which of the following statement?

a. Socket st = new Socket(ServerName, port);

b. Socket st = serverSocket.accept();

c. Socket st = serverSocket.getSocket();

d. Socket st = new Socket(ServerName);

3. To connect to a server running on the same machine with the client, which of the following can't be used for the hostname?

a. “localhost”

b. "127.0.0.1"

c. InetAddress.getLocalHost()

d. "127.0.0.0"

4. To create an InputStream on a socket, say st, which of the following statement is necessary?

a. InputStream ins = new InputStream(st);

b. InputStream ins = st.getInputStream();

c. InputStream ins = st.obtainInputStream();

d. InputStream ins = st.getStream();

5. Which classes are used for connection-oriented socket programming?

a. Socket

b. ServerSocket

c. Both a & b

d. None of the above

3.6 Advantages and Disadvantages of Java Socket Programming

Java Socket programming has been used in a wide range of applications across the world over the years. It bears following advantages and disadvantages:

No	Advantages	Disadvantages
1	Socket programming is flexible & powerful to implement.	It increases complexity, cost and high-Security restrictions.
2	Efficient socket based programming can be easily implemented for general communications to send data in byte and message streams.	Socket-based communications allow only to send packets of raw data between applications.
3	Updated information will be send only between connected devices.	Communication can be established with the machine requested not with another machine.
4	If it is implemented efficiently then it causes Low network traffic.	Both ends i.e. Source and Destination should have the ability to intercept the data.

3.7 Let Us Sum Up

In this unit we have learnt that Java Networking connects two or more computing devices together in order to share resources. Java Sockets are a powerful tool for creating network-based applications in the Java programming language. Java has easy-to-use built-in networking API for network programming. It allows communicating using TCP/IP sockets or UDP sockets over the internet. Java Sockets offer many advantages, such as platform independence, ease of use, scalability and built-in support for secure communication. User Datagram Protocol is connectionless protocol above IP that provides unreliable packet delivery and provides user-level access to low-level IP hardware.

3.8 Answer for Check Your Progress

Check your progress 1: 1. c 2. a 3. a 4. c
Check your progress 2: 1. b 2. a 3. d 4. b 5. c

3.9 Glossary

1. **DHCP**: Dynamic Host Configuration Protocol (DHCP) is a network protocol used to automate the process of configuring devices on IP networks, allowing them to use

network services like DNS and any communication protocol based on UDP or TCP. A DHCP server dynamically assigns an IP address and other network configuration parameters to each device on a network so they can communicate with other IP networks. .

2. **SMTP**: The Simple Mail Transfer Protocol (SMTP) is an Internet standard communication protocol for sending and receiving electronic mail (e-mail).
3. **Latency**: Network latency is the delay in network communication. It shows the time that data takes to transfer across the network. Low latency is always accepted and associated with a positive user experience.
4. **FTP:** File Transfer Protocol allows users to transfer files from one machine to another.
5. **HTTPS**: HTTPS is abbreviated as Hyper Text Transfer Protocol Secure is a standard protocol to secure the communication among two computers one using the browser and other fetching data from web server.
6. **Telnet:** Telnet is a collection of rules designed for connecting one system with another. The connection process here is referred as remote login. The system which requests for connection is the local computer, while the system which accepts the connection is the remote computer.

3.10 Assignment

1. Explain various terminology used in Java Networking Programming.
2. Explain various Java Networking Protocols.
3. Discuss the role of ServerSocket and Socket class in network programming.

3.11 Activities

Develop UDP Client and Server Program.

3.12 Case Study

Explore the services of server by developing Client Server applications.

3.13 Further Readings

- Java: The Complete Reference, Eleventh Edition by Herbert Schildt
- https://www.edureka.co/blog/java-networking/
- https://www.tutorialspoint.com/java/java_networking.htm
- https://www.studytonight.com/java/networking-in-java.php

UNIT 4: JAVA.NET PACKAGE

Unit Structure

4.0 Learning Objectives

After learning this Unit, you will be able to:

- List and define various Networking classes
- List and define various Networking interfaces
- Handle various Socket related exceptions

4.1 Introduction

Java.net is a package that provides a powerful infrastructure in the form of classes as well as interfaces for implementing networking applications. This package can be roughly divided into Low Level API and High Level API. Low Level API deals with Addresses, Sockets and Interfaces while High Level API deals with the URIs, URLs and Connections. As already discussed in previous unit, the java.net package is helpful in Java networking. It supports two protocols i.e. TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). In order to perform several operations on these protocols, java.net package provides various classes and interfaces. In the below sections, the different classes, interfaces and exceptions of the java.net package are discussed.

4.2 Networking Classes

The java.net package provides the classes and interfaces required for network programming. Some of the important classes are listed in following table 1.

Class Name	Description
Authenticator	This class is very essential to get or fetch the connection authentication for networking applications in a network. Authenticator class objects help in getting this authentication.
CacheRequest	This class helps in storing the ResponseCache resources.
CacheResponse	This class helps in retrieving the ResponseCache resources.
Socket	This class implements the sockets that are associated with the client-side.
ServerSocket	This class helps in server socket implementation.

ContentHandler CookieHandler	ContentHandler class is the superclass of the particular classes which are chosen for reading the URL connection objects. This class object offers a callback mechanism to HTTP protocol handler with the HTTP state management policy implementation.
CookieManager	This class extends or implements the CookieHandler class.
DatagramSocket	In a network, to transfer datagram packets, an entity socket is required.
DatagramPacket	DatagramPacket class renders the datagram packet.
InetAddress	This class represents the IP Address.
DatagramSocketImpl	This class helps as a parent class in the implementation of sockets as well as a datagram.
InterfaceAddress	This class represents the network interface address.
JarURLConnection	This class does URL connection establishment to the JAR files.
MulticastSocket	This class helps to Multicast IP packets transfer.
Inet4Address	This class represents the IP version 4 Address.
Inet6Address	This class represents the IP version 6 Address.
HttpURLConnection	This class helps in getting URL connections that have HTTP features.
HttpCookie	This class helps in representing HTTP cookies which carries state information between client-server.
NetPermission	This class offers several network-related permissions.
networkInterface	This class helps in representing the network interface.
PasswordAuthentication	This class acts as a data holder by the authenticator.
Proxy	This class helps in proxy related settings.
ProxySelector	This class helps in proxy server selection.
ResponseCache	This class represents the caches that are associated with the URL

	connections.
SocketPermission	This class does network access provision by using sockets.
URI	This class represents a Uniform Resource Identifier.
URL	This class represents Uniform Resource Locator.
URLConnection	This class acts as a parent class for the classes that communicate between the application & a URL.
URLClassLoader	Loading of certain classes and resources mentioned with the search path is done using this class.
URLDecoder	The decoding of HTML forms are done by using this class.
URLEncoder	The encoding of HTML forms are done by using this class.
URLStreamHandler	This is a class that acts as a parent class for Stream Protocol Handlers.

Table 1. Java.net package classes

InetAddress Class

This class represents the IP Address. It is denoted as public final class InetAddress extends Object implements Serializable. Various important methods of the InetAddress class are listed in table 2.

Method	**Description**
Boolean equals(Object obj)	It compares this object against the specified object.
byte[] getAddress()	It returns the raw IP address of this InetAddress object.
String getHostAddress()	It returns the IP address string in textual presentation form.
Int hashCode()	It returns a hashcode for this IP address.
Boolean isAnyLocalAddress()	It is a utility routine to check if the InetAddress is a wildcard address.
Boolean isLinkLocalAddress()	It is a utility routine to check if the InetAddress is an link local address.
Boolean isLoopbackAddress()	It is a utility routine to check if the InetAddress is a loopback

	address.
Boolean isMCGlobal()	It is a utility routine to check if the multicast address has global scope.
Boolean isMCLinkLocal()	It is a utility routine to check if the multicast address has link scope.
Boolean isMCNodeLocal()	It is a utility routine to check if the multicast address has node scope.
Boolean isMCOrgLocal()	It is a utility routine to check if the multicast address has organization scope.
Boolean isMCSiteLocal()	It is a utility routine to check if the multicast address has site scope.
Boolean isMulticastAddress()	It is a utility routine to check if the InetAddress is an IP multicast address.
Boolean isSiteLocalAddress()	It is a utility routine to check if the InetAddress is a site local address.

Table 2. InetAddress methods

Example: **InetAddressExample.java**

In the following example we have used various methods of INetAddress class.

```
import java.io.*;
import java.net.*;
public class InetAddressExample
{
        public static void main(String[] args)
        {
                try
                {
                InetAddress ip=InetAddress.getByName("www.baou.edu.in");
                System.out.println("Host Name is: "+ip.getHostName());
                System.out.println("\nIP Address is: "+ip.getHostAddress());
        System.out.print("\nisAnyLocalAddress : " +ip.isAnyLocalAddress());
        System.out.print("\nisLinkLocalAddress : " +ip.isLinkLocalAddress());
        System.out.print("\nisLoopbackAddress : " +ip.isLoopbackAddress());
        System.out.print("\nisMCGlobal : " +ip.isMCGlobal());
        System.out.print("\nisMCLinkLocal : " +ip.isMCLinkLocal());
```

```
        System.out.print("\nisMCNodeLocal : " +ip.isMCNodeLocal());
        System.out.print("\nisMCOrgLocal : " +ip.isMCOrgLocal());
                System.out.print("\nisMCSiteLocal : " +ip.isMCSiteLocal());
        System.out.print("\nisMulticastAddress : " +ip.isMulticastAddress());
        System.out.print("\nisSiteLocalAddress : " +ip.isSiteLocalAddress());
        System.out.print("\nhashCode is: " +ip.hashCode());
        }
        catch(Exception e)
        {
                System.out.println(e);
        }
}
}
```

After successful compilation, it will have following output when we execute above program:

```
E:\Adv. Java book 2023\Block 1\Network Program>javac InetAddressExample.java

E:\Adv. Java book 2023\Block 1\Network Program>java InetAddressExample
Host Name is: www.baou.edu.in

IP Address is: 203.77.200.33

isAnyLocalAddress : false
isLinkLocalAddress : false
isLoopbackAddress : false
isMCGlobal : false
isMCLinkLocal : false
isMCNodeLocal : false
isMCOrgLocal : false
isMCSiteLocal : false
isMulticastAddress : false
isSiteLocalAddress : false
hashCode is: -884094943
E:\Adv. Java book 2023\Block 1\Network Program>
```

Figure 1. INetAddress Class Example

Socket Class

This class implements the sockets that are associated with the client-side. Class Declaration is public abstract class Socket extends Object implements Cloaseable. Every server or programs executes on the different systems that has a socket and is bound to the specific port number. Socket provides an endpoint of two way communication link using TCP protocol. Java socket can be connection oriented or connection less. The java.net.Socket class is used to create a socket so that both the client and server can communicate with each other easily.

Various constructors of the Socket class are listed in table 3.

Constructor	Description
Socket()	It creates an unconnected socket, with the system-default type of SocketImpl.
public Socket(InetAddress address, int port)	It creates a stream socket with specified IP address to the specified port number.
public Socket(InetAddress host, int port, boolean stream)	It uses the DatagramSocket.
public Socket(InetAddress address, int port, InetAddress localAddr, int local port)	It creates a connection with specified remote address and remote port.
public Socket(Proxy, proxy)	It creates a connectionless socket specifying the type of proxy.
protected Socket(SocketImpl impl)	It creates a connectionless Socket with a user-specified SocketImpl.

Table 3. Socket constructors

Various important methods of the Socket class are listed in table 4.

.

Method	Description
public InputStream getInputStream()	It returns the InputStream attached with this socket.
public OutputStream getOutputStream()	It returns the OutputStream attached with this socket.
public synchronized void close()	It closes this socket
InetAddress getInetAddress()	It returns the InetAddress that is associated with the socket object.
int getPort()	It returns the port number on which the socket is connected
int getLocalPort()	It returns the local port number on which the socket is created
boolean isBound()	It returns the binding state of the socket.
boolean isClosed()	It returns the closed state of the socket.
boolean isConnected()	It returns the connection state of the socket.

Table 4. Socket methods

ServerSocket Class

This class helps in server socket implementation. Class Declaration is public abstract class ServerSocket extends Object implements Cloaseable. Socket class is used to create socket

and send the request to the server. Java ServerSocket class waits for a request to come over the network. It works on the basis of request and then returns a result to the request.

Various constructors of the ServerSocket class are listed in table 5.

Constructor	**Description**
ServerSocket()	It creates an unbound server socket.
ServerSocket(int port)	It creates a server socket, bound to the specified port.
ServerSocket(int port, int backlog)	It creates a server socket, bound to the specified port, with specified local port.
ServerSocket(int port, int backlog, inetAddress bindAddrs)	It creates a server socket, bound to specified port, listen backlog, and IP address.

Table 5. ServerSocket constructors

Various important methods of the ServerSocket class are listed in table 6.

Method	**Description**
public Socket accept()	It returns the socket and establishes a connection between server and client.
public synchronized void close()	It closes the server socket.
int getLocalPort()	It returns the port number on which the server socket is listening.
void bind(SocketAddress endpoint, int backlog)	It binds the ServerSocket to a specific address (IP address and port number).
void bind(SocketAddress endpoint)	It binds the ServerSocket to a specific address (IP address and port number).
InetAddress getInetAddress()	It returns the local address of the ServerSocket.
boolean isBound()	It returns the binding state of the ServerSocket.
boolean isClosed()	It returns the closed state of the ServerSocket.

Table 6. ServerSocket methods

Example: Socket and ServerSocket class example is discussed in previous unit.

DatagramPacket and DatagramSocket are the two main classes that are used to implement a UDP client/server application. DatagramPacket is a data container and DatagramSocket is a mechanism to send and receive DatagramPackets.

DatagramPacket

In UDP's terms, data transferred is encapsulated in a unit called datagram. A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and

content are not guaranteed. The java.net.DatagramPacket class represents a datagram packet. They are used to implement a connectionless packet delivery service. We can create a DatagramPacket object by using one of the following constructors:

Various constructors of the DatagramPacket class are listed in table 7.

Constructor	**Description**
DatagramPacket(byte[ ] buf, int length)	It constructs a DatagramPacket for receiving packets of length.
DatagramPacket(byte[ ] buf, int length, InetAddress address, int port)	It constructs a datagram packet for sending packets of length to the specified port number on the specified host.
DatagramPacket(byte[ ] buf, int offset, int length)	It constructs a DatagramPacket for receiving packets of length, specifying an offset into the buffer.
DatagramPacket(byte[ ] buf, int offset, int length, InetAddress address, int port)	It constructs a datagram packet for sending packets of length with offset to the specified port number on the specified host.
DatagramPacket(byte[ ] buf, int offset, int length, SocketAddress address)	It constructs a datagram packet for sending packets of length with offset to the specified port number on the specified host.
DatagramPacket(byte[ ] buf, int length, SocketAddress address)	It constructs a datagram packet for sending packets of length to the specified port number on the specified host.

Table 7. DatagramPacket class constructors

Various important methods of the DatagramPacket class are listed in table 8.

Method	**Description**
InetAddress getAddress()	This method returns the IP address of the machine to which this datagram is being sent or from which the datagram was received.

byte[ ] getData()	This method returns the data buffer.
int getLength()	This method returns the length of the data to be sent or the length of the data received.
int getOffset()	This method returns the offset of the data to be sent or the offset of the data received.
int getPort()	This method returns the port number on the remote host to which this datagram is being sent or from which the datagram was received.
SocketAddress getSocketAddress()	This method gets the SocketAddress (usually IP address + port number) of the remote host that this packet is being sent to or is coming from.
void setAddress(InetAddress iaddr)	This method sets the IP address of the machine to which this datagram is being sent.
void setData(byte[] buf)	This method sets the data buffer for this packet.
void setData(byte[ ] buf, int offset, int length)	This method sets the data buffer for this packet.
void setLength(int length)	This method sets the length for this packet.
void setPort(int iport)	This method sets the port number on the remote host to which this datagram is being sent.
void setSocketAddress(SocketAddress address)	This method sets the SocketAddress (usually IP address + port number) of the remote host to which this datagram is being sent.

Table 8. DatagramPacket class methods

DatagramSocket

We use DatagramSocket to send and receive DatagramPackets. DatagramSocket represents a UDP connection between two computers in a network. We use DatagramSocket for both client and server. There are no separate classes for client and server like TCP sockets.

So we can create a DatagramSocket object to establish a UDP connection for sending and receiving datagram, by using one of the following constructors:

Various constructors of the DatagramSocket class are listed in table 9.

Constructor	**Description**
DatagramSocket()	It constructs a datagram socket and binds it to any available port on the local host machine.

DatagramSocket(int port)	It Constructs a datagram socket and binds it to the specified port on the local host machine.
DatagramSocket(int port, InetAddress laddr)	It creates a datagram socket, bound to the specified local address.

Table 9. DatagramSocket class Constructors

These constructors can throw SocketException if the socket could not be opened, or the socket could not bind to the specified port or address. So we have to catch or re-throw this checked exception.

Various important methods of the DatagramSocket class are listed in table 10.

Method	**Description**
Void close()	It closes this datagram socket.
InetAddress getLocalAddress()	It gets the local address to which the socket is bound.
Int getLocalPort()	It returns the port number on the local host to which this socket is bound.
Int getReceiveBufferSize()	It get value of the SO_RCVBUF option for this socket, that is the buffer size used by the platform for input on the this Socket.
Int getSendBufferSize()	It get value of the SO_SNDBUF option for this socket, that is the buffer size used by the platform for output on the this Socket.
Int getSoTimeout()	It retrieve setting for SO_TIMEOUT. 0 returns implies that the option is disabled (i.e.
void receive(DatagramPacket p)	It receives a datagram packet from this socket.
void send(DatagramPacket p)	It sends a datagram packet from this socket.
void setReceiveBufferSize(int size)	It sets the SO_RCVBUF option to the specified value for this DatagramSocket.
void setSendBufferSize(int size)	It sets the SO_SNDBUF option to the specified value for this DatagramSocket.
void setSoTimeout(int timeout)	It enable/disable SO_TIMEOUT with the specified

	timeout, in milliseconds.

Table 10. DatagramSocket class methods

These methods may throw Exception like IOException, PortUnreachableException, SocketTimeoutException. So we have to catch or re-throw them.

Let's see a simple Java program where client sends a text message and server reads it and then prints it.

datagramReceiver.java

```
// program to receive datagram packets using DatagramSocket class
import java.net.*;
public class datagramReceiver
{
	public static void main(String[] args) throws Exception
	{
		DatagramSocket dgsocket = new DatagramSocket(2000);
		byte[] buff = new byte[1024];
		DatagramPacket dgpacket = new DatagramPacket(buff, 1024);
		dgsocket.receive(dgpacket);
		String str = new String(dgpacket.getData(), 0, dgpacket.getLength());
		System.out.println(str);
		dgsocket.close();
	}
}
```

datagramSender.java

```
//program to send datagram packets using DatagramSocket class
import java.net.*;
public class datagramSender
{
	public static void main(String[] args) throws Exception
	{
		DatagramSocket dgsocket = new DatagramSocket();
		String str = "Welcome to BAOU @ Ahmedabad-2023";
		InetAddress ipaddr = InetAddress.getByName("127.0.0.1");
DatagramPacket dgpacket = new DatagramPacket(str.getBytes(), str.length(), ipaddr, 2000);
		dgsocket.send(dgpacket);
		dgsocket.close();
	}
}
```

After successfully compiling both the programs, first run the server program. It waits until client request comes. As soon as client sends requests, server will listen, read and print the data received from client. Following figure shows it in sequence.

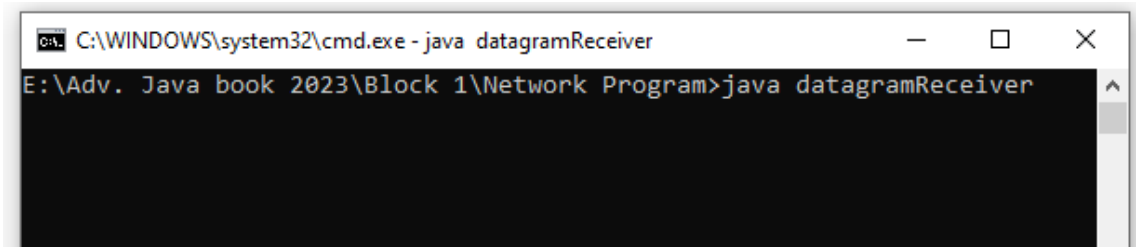


Figure 1. Running Receiver program

Now, run the client program from another prompt as shown in figure 2:

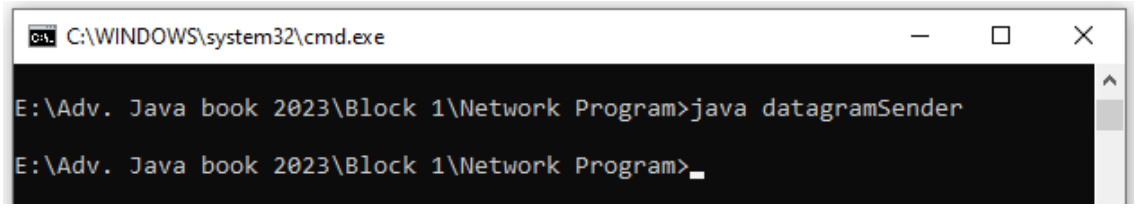


Figure 2. Running Sender program

As soon as the client sends the requests, server receives it and displays the message as shown in figure 3.

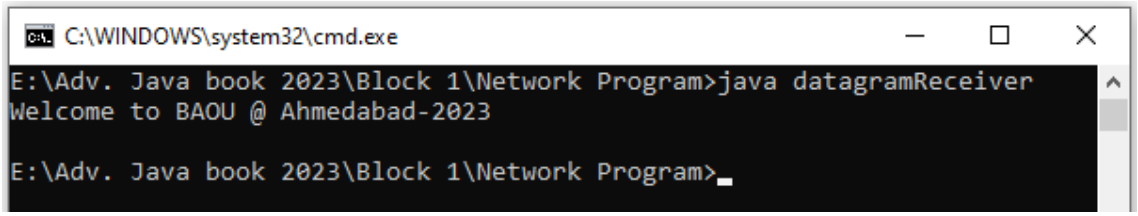


Figure 3. Server program with clients message

URLConnection Class

The URLConnection class is used for accessing the attribute of remote resource. URLConnection is the superclass of all the classes that represent a communication link between application and a URL.

Various important methods of the URLConnection class are listed in table 11.

Methods	**Description**
Object getContent()	It retrieves the contents of this URL connection.
int getContentLength()	It returns the size in byte of content associated with resource.
String getContentType()	It returns type of content found in the resource. If the content is not available, it returns null.
long getDate()	It returns the time and date of the response.
long getExpiration()	It returns the expiry time and date of the resource. If the expiry date is unavailable, it return zero.
long getLastModified()	It returns the time and date of the last modification of the resource.

InputStream getInputStream() throws IOException()	It returns an InputStream that is linked to the resource.
String getRequestProperty(String key)	It returns the value of the named general request property for the given connection.
abstract void connect()	It opens a communications link to the resource referenced by this URL, if such a connection has not already been established.
long getLastModified()	It returns the value of the last-modified header field.
Permission getPermission()	It returns a permission object representing the permission necessary to make the connection represented by this object.
URL getURL()	It returns the value of this URLConnection's URL field.
InputStream getInputStream()	It returns an input stream that reads from this open connection.
OutputStream getOutputStream()	It returns an output stream that writes to this connection.

Table 11. URLConnection class methods

Let's see a simple java program where the URL variable is created to add the specific website / blog URL using the URL command. Then URLConnection is used to open a connection to the above-mentioned URL. Then Map is used to get all fields map of the specific HTTP header. To print all the fields of website URL and their values for loop is used.

URLConnclass.java

```
import java.net.*;
import java.util.*;
public class URLConnclass
{
	public static void main(String[] args)
	{
		try
		{
			URL url = new URL("https://www.baou.edu.in");
			URLConnection urlcon = url.openConnection();
			Map<String, List<String>> header = urlcon.getHeaderFields();
			for (Map.Entry<String, List<String>> mp : header.entrySet())
			{
				System.out.print(mp.getKey() + " : ");
				System.out.println(mp.getValue().toString());
			}
		}
		catch (Exception e1)
		{
			System.out.println(e1);
		}
	}
```

```
}
```

After successfully compiling above programs it will display following output.

```
E:\Adv. Java book 2023\Block 1\Network Program>javac URLConnclass.java

E:\Adv. Java book 2023\Block 1\Network Program>java URLConnclass
X-Frame-Options : [SAMEORIGIN]
null : [HTTP/1.1 200 OK]
Cache-Control : [private]
Set-Cookie : [ASP.NET_SessionId=i5bv1k1q4p0gysaa5lqdyu4y; path=/; HttpOnly; SameSite=Lax]

Content-Length : [75128]
Date : [Mon, 19 Jun 2023 08:13:23 GMT]
Content-Type : [text/html; charset=utf-8]

E:\Adv. Java book 2023\Block 1\Network Program>
```

Figure 4. URLConnection Class

MulticastSocket

In broadcasting packets are sent to all nodes in the network, irrespective of whether they are interested in receiving the communication or not. This leads to wastage of resources. While in Multicasting packets are sent to only those consumers who are interested. Multicasting is based on a group membership concept, where a multicast address represents each group. MulticastSocket is used to receive packets sent to a multicast IP.

Various important Constructor of MulticastSocket class are listed in table 12.

Constructors	**Description**
public MulticastSocket()	It creates a multicast socket. Using this constructor, we have to explicitly set all the fields such as group address, port number etc.
public MulticastSocket(int portnum)	It creates a multicast socket bound on the port specified.
public MulticastSocket(SocketAddress bindsocketaddr)	It creates a multicast socket and binds it to specified socket address. It will create an unbound socket if address is null.

Table 12. MulticastSocket class Constructor

Various important methods of MulticastSocket class are listed in table 13.

Methods	**Description**
public InetAddress getInterface() throws SocketException	It returns the address of the network interface used for outgoing multicast packets.
getTTL public byte getTTL() throws IOException	It returns The time-to-live (TTL) value for this socket.
public void joinGroup(InetAddress	This method is used to join a multicast group.

mcstaddr) throws IOException	
public void leaveGroup(InetAddress mcstaddr) throws IOException	This method is used to leave a multicast group.
public synchronized void send(DatagramPacket dp, byte bttl) throws IOException	This method sends a packet from this socket using the given TTL value. The packet data, packet length, destination address and destination port number are specified by the given DatagramPacket.
public void setInterface(InetAddress inadd) throws SocketException	This method is used to set the address that is used for outgoing multicast packets.
public void setTTL(byte bttl) throws IOException	This method is used to set the TTL value of the socket. The TTL value is the number of hops an outgoing packet can traverse before it is discarded.

Table 13. MulticastSocket class methods

Let's see a simple Java program where client receives a text message when server multicasts it and then prints it.

MultiCastClient.java

```
import java.net.*;
import java.util.*;
import java.io.*;
public class MultiCastClient {

    MulticastSocket Multisocket;

   public MultiCastClient(String ip, int port) throws IOException {
            // important that this is a multicast socket
      Multisocket = new MulticastSocket(port);
      // join by ip
      Multisocket.joinGroup(InetAddress.getByName(ip));
   }

   public void display() throws IOException{
      // make datagram packet to recieve
      byte[] message = new byte[256];
      DatagramPacket packet = new DatagramPacket(message, message.length);

      // recieve the packet
      Multisocket.receive(packet);
```

```
        System.out.println(new String(packet.getData()));
    }

    public void close(){
        Multisocket.close();
    }
            public static void main(String[] args) {
                    try {
                             String ip = args[0];
                             int port = Integer.parseInt(args[1]);
                            MultiCastClient Multiclient = new MultiCastClient(ip, port);
                            Multiclient.display();
                            Multiclient.close();
                    } catch (IOException ex) {
                            ex.printStackTrace();
                    }
            }
}
```

MultiCastServer.java

```
import java.net.*;
import java.util.*;
import java.io.*;
public class MultiCastServer
{
    DatagramSocket dgserverSocket;
    String ip;
    int port;
    public MultiCastServer(String ip, int port) throws SocketException, IOException{
        this.ip = ip;
        this.port = port;
        // socket used to send
        dgserverSocket = new DatagramSocket();
    }

    public void send() throws IOException{
        // make datagram packet
        byte[] message = ("Welcome to BAOU @ Ahmedabad").getBytes();
        DatagramPacket packet = new DatagramPacket(message, message.length,
            InetAddress.getByName(ip), port);
```

```
    // send a packet
    dgserverSocket.send(packet);
  }

  public void close(){
    dgserverSocket.close();
  }
      public static void main(String[] args)
      {
            try {
                  String ip = args[0];
                  int port = Integer.parseInt(args[1]);
                  MultiCastServer server = new MultiCastServer(ip, port);
                  server.send();
                  System.out.println("Sent a  multicast message.");
                  server.close();
            }
            catch (IOException ex)
            {
                  ex.printStackTrace();
            }
      }
}
```

Here, the client must subscribe to the IP before it can start receiving any packets. If you start the server and call the send() method, and then make a client (call display()) then nothing will happen because the client connected after the message was sent.

After successfully compiling and running both the programs with specific IP address and port number the following output will be displayed in sequence.

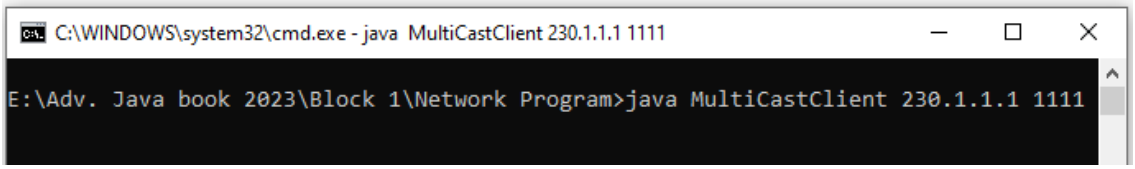


Figure 5. Running Client program

Now, run the client program from another prompt as shown in figure 2:

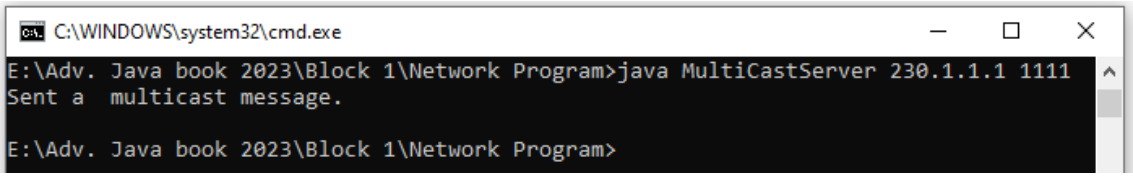


Figure 6. Running Server program

As soon as the client sends the requests, server receives it and displays the message as shown in figure 3.

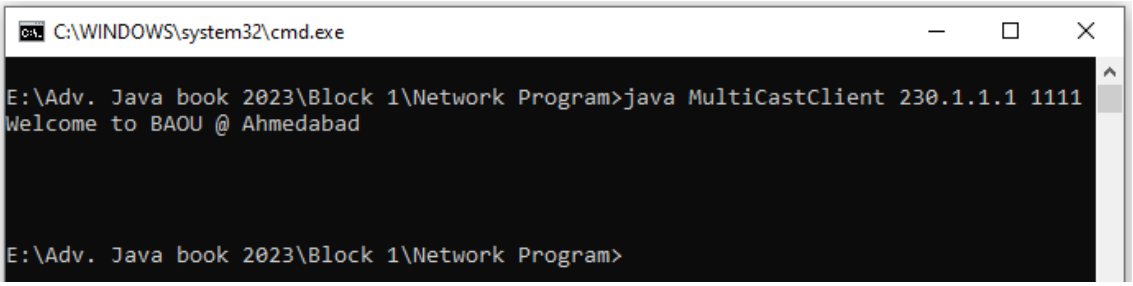


Figure 7. Client program with Server message

Check your progress 1

1. Which constructor of DatagramSocket class is used to creates a datagram socket and binds it with the given Port Number?

a. DatagramSocket(int port)

b. DatagramSocket()

c. DatagramSocket(int port, InetAddress address)

d. None of the above

2. Which of these class is used to encapsulate IP address and DNS?

a. DatagramPacket

b. URL

c. InetAddress

d. ContentHandler

3. In InetAddress class which method it returns the host name of the IP Address?

a. public String getHostName()

b. public String getHostAddress()

c. public static InetAddress getLocalHost()

d. None of the above

4. Which classes are used for connection-less socket programming?

a. DatagramSocket

b. DatagramPacket

c. Both a & b

d. None of the above

5. The URLConnection class can be used to read and write data to the specified resource referred by the URL?

a. True

b. False

6. The java.net.InetAddress class represents an?

a. MAC Address

b. IP Address

c. Protocol

d. Socket

4.3 Networking Interfaces and Exceptions

Some important interfaces in the java.net package are:

1. ContentHandlerFactory

The ContentHandlerFactory interface provides a method that creates and returns an appropriate ContentHandler object for a given MIME type.

2. CookiePolicy

The java.net package's CookiePolicy interface provides various classes for implementing various networking applications. It decides which cookies are accepted and which are rejected. There are 3 pre-defined policy implementations in CookiePolicy:

- ACCEPT_ALL
- ACCEPT_NONE
- ACCEPT_ORIGINAL_SERVER.

3. CookieStore

A CookieStore is an interface that specifies a cookie storage space. CookieManager adds cookies to the CookieStore with each HTTP response and retrieves cookies from the CookieStore with each HTTP request.

4. DatagramSocketImplFactory

This interface illustrates a factory for datagram socket implementations. It is used by the classes DatagramSocket to create actual socket implementations.

5. FileNameMap

The FileNameMap interface is a simple interface that implements a tool for highlighting a file name and a MIME type string. FileNameMap reads a data file and charges a filename map.

6. ProtocolFamily

This interface defines a communication protocol family. The ProtocolFamily interface includes a name() method that returns the protocol family's name.

7. SocketImplFactory

The SocketImplFactory interface defines a SocketImpl instance factory. The socket class uses it to create socket implementations that implement several policies.

8. SocketOption

The SocketOption interface allows users to control how the socket will behave. It is frequently necessary to develop necessary features in Sockets. SocketOption allows the user to configure a variety of standard options.

9. URLStreamHandlerFactory

The URLStreamHandlerFactory interface defines a method that creates a URLStreamHandler object for a specific protocol. The interface is implemented by classes that select URLStreamHandler subclasses to process particular protocol types.

Exceptions:

Some important exceptions needs to be taken care while working with Socket are:

1. SocketException

The java.net.SocketException represents a generic socket error, which can represent a range of specific error conditions i.e the subclasses listed below.

2. BindException

The java.net.BindException represents failure to bind a socket to a local port. The most common reason for this is that the local port is already in use.

3. ConnectException

The java.net.ConnectException occurs when a socket can't connect to a specific remote host and port. The reasons behind this will be the remote server does not have a service bound to that port, or that it is so swamped by queued connections, it cannot accept any further ones.

4. NoRouteToHostException

The java.net.NoRouteToHostException is thrown when, due to a network error, it is impossible to find a route to the remote host. The reason for this may be local (i.e., the network on which the application is running), a temporary gateway or router problem or the fault of the remote network to which the socket is trying to connect. Second common reason for this is that firewalls and routers are blocking the client software.

5. InterruptedIOException

The java.net.InterruptedIOException occurs when a read operation is blocked for sufficient time to cause a network timeout.

Check your progress 2

1. In CookiePolicy, which of the following is not pre-defined policy implementations.

a. ACCEPT_ALL

b. ACCEPT_NONE

c. ACCEPT_ALL_ORIGINAL_SERVER

d. ACCEPT_ORIGINAL_SERVER

2. The SocketException is an exception in Java that is thrown to indicate that an error was encountered while creating or accessing a Socket.

a. true

b. false

3. A BindException is thrown if you try to construct a Socket or ServerSocket object on a local port that is in use.

a. true

b. false

4.4 Let Us Sum Up

In this unit we have discussed important classes and interfaces of java.net package. If user wants to work with URL manipulation then there are classes like URI, URl and URLConnection. We also discussed that if user wants to communicate data using UDP protocvol then there are DatagramSocket and DatagramPacket classes are there. The HttpCookie class helps to carry state information between client-server. Apart from this there are various classes and interfaces which user can use as per their requirements.

4.5 Answer for Check Your Progress

Check your progress 1: 1. c 2. c 3. a 4. c 5. a 6. b
Check your progress 2: 1. c 2. a 3. a

4.6 Glossary

1. **Addresses**: It is an IP addresses as a numerical label assigned to each device connected to a computer network that uses the Internet Protocol version 4 (32 bits) or version 6 (128 bits).
2. **Sockets**: It is an abstraction of a bi-directional communication channel between hosts, input and output streams are used to send and receive data.
3. **Interfaces**: It is a point of interconnection between a computer and a network. It can be Hardware Interface i.e Network Interface Card and Software Interface i.e Loopback Interface (lo0).
4. **URI**: It is a Uniform Resource Identifier, a sequence of characters that identifies a logical or physical resource.
5. **URL**: It is a Uniform Resource Locator, a reference to a web resource that specifies its location on a network, a special kind of URI. For example: https://www.google.com/
6. **Connection**: It represents a connections to the resource pointed to by URLs.

4.7 Assignment

1. Discuss the role of Authenticator, CookieManager and InterfaceAddress classes of java.net package in detail.
2. Discuss about Proxy and ProxySelector class

4.8 Activities

Develop a program to multicast the message using java.net package.

4.9 Case Study

Explore different URL related classes of java.net package.

4.10 Further Readings

- https://docs.oracle.com/javase/8/docs/api/java/net/package-summary.html
- https://www.codejava.net/java-se/networking/
- https://developer.android.com/reference/java/net/Socket

BLOCK 2: JDBC, Stored Procedure and Functions

Block Introduction

Through Java Database Connectivity user can develop a database application and manipulate the records of the database. JDBC provides a standard Java API for database-independent connectivity between the java program and a wide range of databases.

In this block, we will discuss about accessing data from database through Java application. Here, we are discussing various types of JDBC drivers used in JDBC applications based on requirements. In this block, students are practically demonstrated the connection establishment with database, accessing data from database using different interfaces and classes of SQL package of JDK. Students were also made clear regarding the exception to be handled while writing the database application. Students will also learn to create and call stored procedure and function practically.

Block Objective:

After learning this block, you will be able to:

- Differentiate various JDBC drivers and their applications
- Explain the various steps of writing JDBC Program
- Use various JDBC Statements
- Define Prepared Statements
- Create and call a Stored Procedure from database
- Create and call a Stored Procedure from database
- Decide which class and their methods to use from java.sql package

Block Structure

Block Summary

In this block, students have learnt and understand about Java Database Connectivity concept to connect a java program with underlying database. This block has explored various functionality of class library in JDK along with querying of database. The students were well explained on the concepts of Java Database Connectivity using ResultSet, Prepared Statement, Callable Statement interface practically. The concept related to stored procedure and functions were also discussed practically with its applications. Students were also made aware about the exception they need to handle while accessing the database.

Block Assignment

Short Answer Questions:

1. Define Database and it component.
2. Differentiate stored procedure and function.
3. Explain the functionality of ResultSet object.
4. What is the use of JDBC DriverManager?
5. Define metadata.

Long Answer Questions:

1. Discuss different steps required for connecting Java application with database.
2. Discuss different JDBC drivers.
3. Discuss various types of Statements.
4. Write a short note on ResultsetMetadata.
5. Explain various methods of statement interface.
6. Write a JDBC program to navigate records of students database.

UNIT 1: INTRODUCTION TO JDBC

Unit Structure

1.0 Learning Objectives

After learning this Unit, user will be:

- Able to define CLI
- Able to explain JDBC architecture
- Able to discuss different JDBC drivers
- Able to establish database connection
- Able to define different JDBC statements
- Able to access data using Result Sets

1.1 Introduction

JDBC is same as ODBC but it is particularly designed for JAVA while ODBC is language independent. Java Database Connectivity (JDBC) is the universally industry accepted standard for database independent connection between Java applications and different SQL databases. All the benefits of java are equally applicable to JDBC. The JDBC API defines various java classes and interfaces that represent database connections, SQL statements, result sets and database metadata. JDBC provides following functionality to a Java programmer:

- To make a connection with a database
- Write SQL statements
- Process the results

The JDBC API is implemented through a driver manager which will support various drivers allowing the user to connect application with different databases. JDBC meta-data access allows the developer to develop sophisticated applications which requires to understand the functionality of the specific database connection. JDBC allows users to reap the benefits of Internet-standard URLs to recognize database connections. JDBC is available anywhere as being a core part of the Java Platform. It means with java applications user can truly write database applications once and manipulate the data anywhere it is required.

1.2 Call Level Interface (CLI)

It was developed in the early 1990's and defined only for the programming languages C and COBOL. Call Level Interface (CLI) is a database programming interface from the SQL Access Group (SAG). CLI is an attempt towards standardizing the SQL language for accessing database. Microsoft's ODBC also follows and works in line to the CLI along with adding its own extensions. Using CLI, SQL statements are directly passed to the server

without recompilation. It is also known as the callable Structured Query Language (SQL) programming interface. Along with being an application programming interface (API), it is a software standard to embed Structured Query Language (SQL) code in a host program. The Call Level Interface defines a mechanism for a program regarding submission of a SQL queries to the database system (DBMS) and accessing the returned records in a consistent manner. CLI supports and encourages a rich set of client - server tools which enable access to databases through dynamic-link libraries (DLL). Best example of the use of the CLI standard is the Open Database Connectivity (ODBC) specification, which allows applications to transparently access different database systems from different vendors. ANSI C, C#, Visual Basic .NET (VB.NET), Java, Pascal and Fortran languages are the examples that support Call Level Interface.

Check your progress 1

1. JDBC stands for?

a. Java Database Connect

b. Java Database Connectivity

c. Json Database Connectivity

d. Json Database Connect

2. What is the full form of CLI?

a. Control Line Interface

b. Central Line Interface

c. Command Line Interface

d. Code Line Interface

1.3 Implementation of JDBC

ODBC is not suitable for a direct use from the Java programming language because it uses a C interface hence platform dependent. The JDBC API was developed after ODBC and being a Java API, it supports a natural java interface for working with SQL. JDBC provides a “pure Java” solution for application development. All modern relational DBMSs (Database Management Systems) support SQL. With the help of JDBC it is possible to write a single database application that can run on different platforms and interact with various underlying DBMSs. The JDBC API includes following objects:

DataSource object: This object is used to establish connections. We can also use the Driver Manager to establish a connection. To establish a connection through a DataSource object is more preferred method.

Connection object: This object handles the connection with the database. To create the statements an application uses the connection object.

Statement: Statement, PreparedStatement and CallableStatement objects are used for executing SQL statements. A PreparedStatement object is used to execute precompiled statements. This object can be executed multiple times with different parameter values provided for each execution. A CallableStatement is used to call stored procedures that return values.

ResultSet: This object contains the results of a SQL query. Whenever a SQL query executed by a statement object it returns a ResultSet to an application. Once the ResultSet is retrieved, user can use various methods of it to iterate through the results of the query.

SQL Exception: This class will help us to handle for any kind of errors that occur in a database application during execution.

Check your progress 2

1. What are the major components of the JDBC?

a. DriverManager, Driver, Connection and Statement

b. DriverManager, Driver, Connection, Statement and ResultSet

c. DriverManager, Statement and ResultSet

d. DriverManager, Connection, Statement and ResultSet

2. Which is responsible for getting a connection to the database?

a. Statement

b. Connection

c. Driver

d. ResultSet

1.4 JDBC Architecture

The JDBC driver manager is an established and reliable way of the JDBC architecture, which specifies objects for connecting Java applications to a JDBC driver. The JDBC architecture supports two-tier and three-tier working models for accessing a database.

1. Two-tier Model (Client-Server)

In this model, a Java applet or application interacts directly to the data source. The JDBC driver enables interaction between the application and the data source. When a user sends a query to the data source, the data in the form of results for these queries are sent back to the user. In this model, the data source used may be located on local machine or on a remote machine on a network to which a user is connected. So, this model is known as a client -

server environment where the user's machine will work as a client and the machine where the data source running will work as a server. The network connection can be either intranet or Internet.

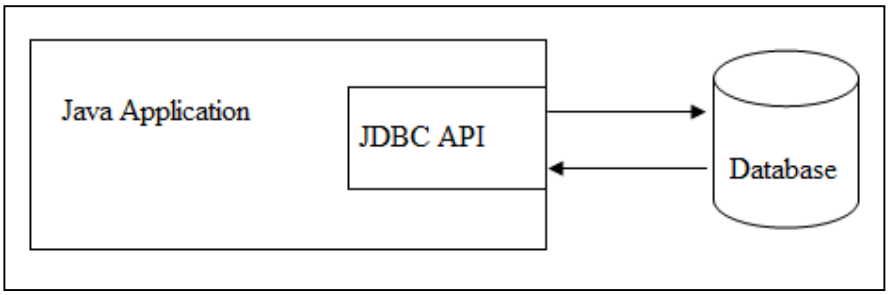


Figure 1: Two-tier Model (Client-Server)

2. Three-tier Model

In this model having one more layer as middle tier where the user's commands or queries are first sent to middle-tier services, which subsequently transmit the commands to the data source. The data source sends the results back to the middle tier, and then the middle tier passes it to the user. In this model application deployment becomes very easy and also provides performance benefits. Generally, the middle tier is developed in C or C++ language. This model is complex and more secured.

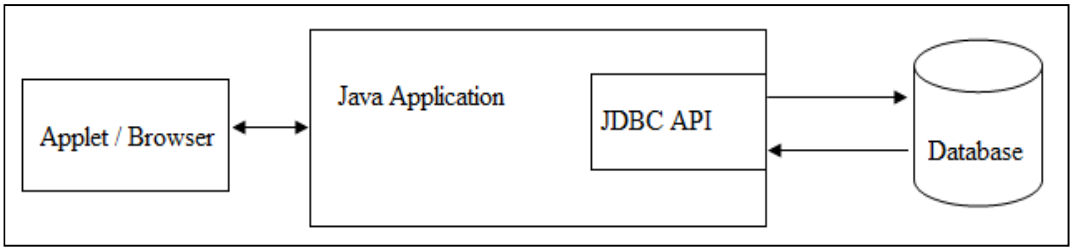


Figure 2: Three-tier Model

Check your progress 3

1. The JDBC API supports _____ processing models for database access

a. Two-tier

b. Three-tier

c. Both a & b

d. None of these

2. JDBC Architecture consists of _____ layers.

a. 1

b. 2

c. 3

d. 4

1.5 JDBC Drivers

Whenever user wants to connect a database from a Java program, it is compulsory to use a JDBC driver made specifically for the database to which user wants to connect. Every vendor provides a custom-built JDBC driver to serialize SQL queries back and forth from the Java application to their specific database. JDBC is a standard specification; a Java program that uses the JDBC API can connect to any database management system (DBMS) for which there is a JDBC driver available. There are 4 types of JDBC drivers:

Type 1: JDBC-ODBC bridge driver

A type 1 JDBC driver consists of a Java part that translates the JDBC interface calls to ODBC calls. An ODBC bridge then calls the ODBC driver of the given database i.e. the driver converts JDBC method calls into ODBC function calls. Sun provides a JDBC-ODBC Bridge driver: sun.jdbc.odbc.JdbcOdbcDriver. This driver is native code and not Java, and is closed source. This JDBC drivers are not recommended for production systems. JDBC-ODBC Bridge driver is not multi threaded. JDBC-ODBC Bridge can open only one Statement object per connection at a time. This type of driver is not considered a deployment-level driver and is generally used for development and testing purposes only.

Note: In Java 8 version, the JDBC-ODBC Bridge has been removed.

Advantage of JDBC-ODBC bridge driver

- This driver is easy to use.
- This driver allows easy connectivity to all database supported by the ODBC Driver.

Disadvantage JDBC-ODBC bridge driver

- The performance gets degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the each client machine.
- Execution time remains slow.
- It is dependent on ODBC Driver.
- This type of driver uses Java Native Interface (JNI) to make ODBC call.

Type 2: Native-API Driver (partially java driver)

A type 2 JDBC driver is like a type 1 driver, except the ODBC part is replaced with a native code part instead. The native code part is targeted at a specific database product i.e. uses the client-side libraries of the database product. The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database native API. It is not written entirely in java. These API are written in C and C++. This driver is used in situations where a type 3 or type 4 driver is not available.

Advantage of Native-API Driver (partially java driver)

- Here the performance gets upgraded compare to JDBC-ODBC bridge driver.
- This driver contains additional features.

Disadvantage of Native-API Driver (partially java driver)

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

Type 3: Network protocol driver (fully java driver)

A type 3 JDBC driver is an all Java driver that sends the JDBC interface calls to an intermediate server. The intermediate server then connects to the database on behalf of the JDBC driver. The middle-tier (application) server translates JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java. If the Java application is accessing multiple types of databases at the same time, type 3 is the best preferred driver.

Advantage Network protocol driver (fully java driver)

- No client side library is required because of application server that can perform many tasks like load balancing, auditing , logging etc.
- Database Independency.
- Provide facility to switch over from one database to another database.

Disadvantage Network protocol driver (fully java driver)

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

Type 4: Pure Java driver (Thin driver)

The JDBC type 4 driver, also known as the Direct to Database Pure Java Driver, is a database driver implementation that converts JDBC calls directly into a vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language. It is

implemented for a specific database product. Today, most JDBC drivers are type 4 drivers. If you are accessing Oracle, Sybase, or IBM type of database then the preferred driver type is 4.

Advantage of Pure Java driver

- It provides better performance than all other drivers.
- No software requirement at client side or server side.
- It does not require any native library.
- It does not require any Middleware server.

Disadvantage of Pure Java driver

- This driver is dependent on the database.

Note: Type 4 driver is fastest JDBC driver written in JAVA.

The following table contains various JDBC supported database drivers.

Relational Database	**Driver Name**
Oracle	com.oracle.jdbc.Driver
MySQL	com.mysql.jdbc.Driver
IBM DB2 App	com.ibm.db2.jdbc.app.DB2Driver
IBM DB2 Net	com.ibm.db2.jdbc.net.DB2Driver
Sybase	com.sybase.jdbc.SybDriver
Microsoft SQL Server	com.microsoft.sqlserver.jdbc.SQLServerDriver
Postgre	org.postgresql.Driver
JDBC-ODBC Bridge	sun.jdbc.odbc.JdbcOdbcDriver
Teradata	com.teradata.jdbc.TeraDriver

Table 1: JDBC Supported Database Drivers

Check your progress 4

1. Mainly how many types of JDBC drivers are there?

a. 1

b. 2

c. 3

d. 4

2. Thin driver is also known as?

a. Type 3 Driver

b. Type-2 Driver

c. Type-1 Driver

d. Type-4 Driver

3. Which JDBC driver can be used in servlet and applet both?

a. Type 3

b. Type 4

c. Type 3 and Type 2

d. Both a & b

4. Which of the following driver is the fastest one?

a. JDBC-ODBC Bridge Driver

b. JDBC Net Pure Java Driver

c. Native API Partly Java Driver

d. Network Protocol Driver

1.6 Let Us Sum Up

In this unit, we have learnt that Java Database Connectivity, is a standard Java API for database-independent connectivity. JDBC is the data access mechanism used in java applications. It is a part of the Java standard edition. JDBC helps the programmer to query the data from the database and to update the data in the database. It is used in relational databases like MySQL, Oracle, PostgreSQL etc. Today, there are various types of JDBC versions available to connect JAVA API to the database. There are total seven types of JDBC version available like JDBC 1.2, JDBC 2.1, JDBC 3.0, JDBC 4.0, JDBC 4.1, JDBC 4.2 and JDBC 4.3. JDBC 4.3 is the latest stable version of JDBC. We have also discussed that JDBC architecture is mainly divided into 5 main components such as DriverManager, Driver, Connection, Statement and ResultSet. Different interfaces and classes of JDBC API are used to establish a connection and interact with databases. The JDBC architecture in Java is mainly categorized in two types i.e. 2-tier model and 3-tier model.

1.7 Answer for Check Your Progress

Check your progress 1: 1. b 2. c
Check your progress 2: 1. b 2. c
Check your progress 3: 1. c 2. b
Check your progress 4: 1. d 2. d 3. d 4. b

1.8 Glossary

1. Java Database Connectivity: JDBC is the Standard Java API for database that involves Java programming language with databases.
2. Driver: This interface helps to handles the communications with the database server.
3. Connection: This interface is used to contact the database with all its methods. The connection object represents communication context, i.e., connection object allows all communication with database.
4. Statement: The objects of this interface are used to submit the SQL statements to the database. Some of the derived interfaces accept parameters in addition to executing stored procedures.
5. ResultSet: This object contains data retrieved from a database after user executes an SQL query using Statement objects. It acts as an iterator to allow user to traverse through its data.
6. SQLException: This class handles any errors that occur in a database application.

1.9 Assignment

1. Discuss various types of JDBC drivers and its applications.
2. Write short note on JDBC Architecture.
3. Explain the concept of Call Level Interface.
4. Discuss various benefits of JDBC.

1.10 Activities

Analyse and differentiate various JDBC drivers.

1.11 Case Study

Differentiate Java Database Connectivity (JDBC) from ODBC.

1.12 Further Readings

- Java: The Complete Reference, Eleventh Edition by Herbert Schildt
- https://docs.oracle.com / javase / tutorial / jdbc / overview / architecture.html
- https://www.herongyang.com / JDBC / Overview-JDBC-Version.html
- https://www.studytonight.com / java / introduction-to-jdbc.php
- https://www.educba.com / jdbc-architecture/

- https://www.progress.com / faqs / datadirect-jdbc-faqs / what-are-the-types-of-jdbc-drivers

UNIT 2: EXPLORING JAVA.SQL PACKAGE

Unit Structure

2.0 Learning Objectives

After learning this Unit, you will be:

- Able to use various classes and interfaces of jav.sql package
- Able to define different JDBC statements
- Able to retrieve data using Result Sets
- Able to define Scrollable and Updatable ResultSet
- Able to define Metadata

2.1 Introduction

JDBC API is divided into two main packages. Whenever we are using JDBC, we need to import these packages to use classes and interfaces in our application. These packages are:

- java.sql
- javax.sql

java.sql package include various classes and interface to perform almost all JDBC operation like creating and executing SQL Queries. The java.sql package contains the API for manipulating data stored in a data source (generally a relational database) using the Java programming language. The javax.sql package is also known as JDBC extension API. It provides various classes and interface to access server-side data. The javax.sql package provides the functionality of DataSource interface as an alternative to the DriverManager for establishing a connection with a data source, Connection pooling and Statement pooling, Distributed transactions and Rowsets. In this unit, we will discuss important classes and interfaces of java.sql package in detail.

2.2 DriverManager Class

DriverManager class works as an intermediary between java application and the drivers of the database, an application want to connect with. The Driver Manager is a very important class that helps in defining an object which connects Java applications to a JDBC driver. DriverManager is the backbone of the JDBC architecture. The main task of the JDBC database driver is to load all the drivers found in the system. The DriverManager also helps in selecting the most suitable driver from the previously loaded drivers when a new open database is connected. This class works between the user and the drivers. The main responsibility of this class is to keep track of the drivers that are available and establishing a connection between a database and the suitable driver. It also keeps records of the driver login time limits and printing of log and tracing messages. This class is mainly helpful for the

simple application. The getConnection() method is the most frequently used method of this class. Various methods of the DriverManager Class are shown in below table 1.

Methods	Details
public static synchronized void registerDriver(Driver driver)	It is used to register the given driver with DriverManager class.
public static synchronized void deregisterDriver(Driver driver)	It is used to drops the driver from the list of drivers registered in the DriverManager class.
public static Connection getConnection(String url) throws SQLException	It tries to establish the connection to a given database URL. This method is used to establish the Connection of single-user database software like microsoft-access.
public static Connection getConnection(String url,String userName,String password) throws SQLException	It tries to establish the connection to a given database URL. It is used to establish the Connection of multiuser database software like Oracle, Sybase.
public static Driver getDriver(String url)	It attempts to locate the driver by the given string.
pubic static int getLoginTimeout()	This method returns the duration of time a driver is allowed to wait in order to establish a connection with the database.
pubic static void setLoginTimeout(int sec)	The method provides the time in seconds. It is the maximum time that a driver is allowed to wait in order to establish a connection with the database.
public static Connection getConnection(String URL, Properties prop) throws SQLException	It tries to establish the connection to a given database URL and its properties.

Table 1: Methods of the DriverManager Class

Code to get the connection with Oracle database using Oracle thin driver:

```
Connection conn = DriverManager.getConnection(
      "jdbc:oracle:thin:@localhost:1521:orcl", "username", "password");
// place your own url, username, password
```

Code to get the connection with MySQL database:

```
Connection conn = DriverManager.getConnection(
     "jdbc:mysql:///dbname","username","password");
// place your own url, username, password
```

Check your progress 1

1. The _____ method returns the object of java.lang.Class object.

a. Class.Name()

b. Class.forAttribute()

c. Class.Attributes()

d. Class.forName()

2. Which of the following method is static and synchronized in JDBC API?

a. prepareCall()

b. getConnection()

c. executeQuery()

d. executeUpdate()

2.3 Connection Interface

A Connection is a session between a Java application and a specific database. This interface is a factory of Statement, PreparedStatement and DatabaseMetaData. This interface provides many methods for transaction management like commit(), rollback(), setAutoCommit() and setTransactionIsolation() etc. A Connection object's getMetaData method provides information regarding its tables, supported SQL grammar, stored procedures, the capabilities of this connection and so on. There are some common constant fields that are present in the Connect interface. These fields are TRANSACTION_NONE, TRANSACTION_READ_COMMITTED, TRANSACTION_READ_UNCOMMITTED, TRANSACTION_REPEATABLE_READ and TRANSACTION_SERIALIZABLE.

Various commonly used methods of Connection interface are listed in below table 2.

Methods	Details
public Statement createStatement()	It creates a statement object that can be used to execute SQL queries.
public Statement createStatement(int resultSetType,int resultSetConcurrency)	Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
public void setAutoCommit(boolean status)	It is used to set the commit s status. By default it is true.
public void commit()	It saves the changes made since the previous commit/rollback permanent.
public void rollback()	It drops all changes made since the previous

	commit/rollback.
public void close()	It closes the connection and releases JDBC resources immediately.

Table 2: Methods of the Connection Interface

2.4 Statement

This interface object is used for executing a static SQL statement and returning the results it generates. Once a connection is established we can interact with the database. By default, at a time only one ResultSet object per Statement object can be open. Therefore, if two ResultSet object is interleaved with the reading of another, each must have been created by different Statement objects. The JDBC Statement, CallableStatement and PreparedStatement interfaces defines various methods and properties to send SQL or PL/SQL commands and receive data from database. A java.sql.Connection object has methods to create a java.sql.Statement object, which can be used to execute SQL queries against a database. Several methods on java.sql.Statement object can be used to execute a SQL statement against the database. The Statement interface provides various important methods as shown in below table 3.

Methods	Description
public boolean execute(String sql)	It executes the given SQL statement, which may return multiple results.
public int[] executeBatch( )	It submits a batch of commands to the database for execution and if all commands execute successfully, returns an array of update counts.
public ResultSet executeQuery( )	It executes the given SQL statement, which returns a single ResultSet object.
Public int executeUpdate(String sql)	It executes the given SQL statement, which may be an INSERT, UPDATE or DELETE statement or an SQL statement that returns nothing, such as an SQL DDL statement.

Table 3: Methods of Statement Interface

Syntax:

Create a statement using connection object:

- Statement statement = connection.createStatement();

Check your progress 2

1. _____ method can be used for any SQL (Select and Update both) statements.

a. executeQuery

b. executeUpdate

c. execute

d. All of the above

2. Which of the following is not a valid statement in JDBC?

a. Statement

b. QueryStatement

c. PreparedStatement

d. CallableStatement

2.5 ResultSet

The type of a ResultSet object determines the level of its functionality in two areas: the ways in which the cursor can be manipulated, and how concurrent changes made to the underlying data source are reflected by the ResultSet object.

The ResultSet interface provides various methods for retrieving and manipulating the results of executed queries. ResultSet objects can have different functionality and characteristics. These characteristics are a type and concurrency.

ResultSet Types:

The type of a ResultSet object determines the level of its functionality in two fields: First, the ways in which the cursor can be manipulated, and the second, how concurrent changes made to the underlying data source are reflected by the ResultSet object. The sensitivity of a ResultSet object is categorized by one of three different ResultSet types:

- TYPE_FORWARD_ONLY

In this type, the result set cannot be scrolled; its cursor will move forward only, from before the first row to after the last row.

- TYPE_SCROLL_INSENSITIVE

In this type, the result can be scrolled; its cursor can move both forward and backward relative to the current position and it can move to an absolute position. The result set is insensitive to updates made to the underlying data source while it is open.

- TYPE_SCROLL_SENSITIVE

In this type, the result can be scrolled; its cursor can move both forward and backward relative to the current position, and it can move to an absolute position. The result set reflects updates made to the underlying data source while the result set is open.

ResultSet Concurrency:

The concurrency of a ResultSet object decides the level of update functionality supported. There are two concurrency levels:

- CONCUR_READ_ONLY:

Here, the ResultSet object cannot be updated using the ResultSet interface.

- CONCUR_UPDATABLE:

Here, the ResultSet object can be updated using the ResultSet interface.

The default ResultSet type is TYPE_FORWARD_ONLY and the default ResultSet concurrency is CONCUR_READ_ONLY.

The following line of code will create a scrollable and read only ResultSet object:

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                     ResultSet.CONCUR_READ_ONLY);
```

The following line of code will create a scrollable and updatable ResultSet object:

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
```

Various important methods of ResultSet interface are detailed in table 4.

Method	Details
public boolean next()	It is used to move the cursor to the one row next from the current position.
public boolean previous()	It is used to move the cursor to the one row previous from the current position.
public boolean first()	It is used to move the cursor to the first row in result set object.
public boolean last()	It is used to move the cursor to the last row in result set object.
public boolean absolute(int row)	It is used to move the cursor to the specified row number in the ResultSet object.
public boolean relative(int row)	It is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
public int getInt(int columnIndex)	It is used to return the data of specified column index of the current row as int.
public int getInt(String columnName)	It is used to return the data of specified column name of the current row as int.
public String getString(int columnIndex)	It is used to return the data of specified column index of the current row as String.
public String getString(String	It is used to return the data of specified column

columnName)	name of the current row as String.

Table 4: Methods of ResultSet interface

Using the Method next:

In order to access the records fetched in resultset object, we will go to each row and retrieve the values according to their types. The method next moves a cursor to the next row and makes that row available to operate. Initially the cursor is positioned just above the first row of a ResultSet object, the first call to the method next moves the cursor to the first row and makes it the current row. Successive invocations of the method next move the cursor down one row at a time from top to bottom.

Using the getXXX Methods:

Here, XXX denotes the types of data we are trying to get from the underlying database. We can use the getXXX method of the appropriate type to retrieve the value in each column. For example, the first column in each row of resultset is studno, which stores a value of SQL type int. We will use the method getInt() for retrieving a value of SQL type integer. The second column in each row stores a value of student name of SQL type varchar, then getString() method will be used to retrieve the values. The following code accesses the values stored in the current row of ResultSet rs and prints a line with the employee number followed by name, department and salary. Each time the method next is invoked, the next row becomes the current row, and the loop continues until there are no more rows in rs.

```
String query = "SELECT * from employee";
ResultSet rs = stmt.executeQuery(query);
while (rs.next())
{
    int num= rs.getInt("empdno");
    String s=rs.getString("empname");
    String dept=rs.getString("empdept");
    float n = rs.getFloat("empsalary");
    System.out.println(num +" " +s + "" + dept +" "+ n);
}
```

Check your progress 3

1. What are the types of ResultSet in JDBC?

a. Forward ResultSet

b. Scrollable ResultSet

c. Only a

d. Both a and b

2. Which of the following is not a type of ResultSet object?

a. CONCUR_WRITE_ONLY

b. TYPE_FORWARD_ONLY

c. TYPE_SCROLL_INSENSITIVE

d. TYPE_SCROLL_SENSITIVE

2.6 Metadata

DatabaseMetaData interface provides methods to get metadata like database product name, database product version, driver name, name of a total number of tables, a name of a total number of views of a database.

This interface is implemented by driver vendors to let users know the capabilities of a Database Management System (DBMS) in combination with the driver based on JDBC technology. DatabaseMetaData interface commonly used methods are listed in table 5.

Method	Details
public String getDriverName() throws SQLException	It returns the name of the JDBC driver.
public String getDriverVersion() throws SQLException	It returns the version number of the JDBC driver.
public String getUserName() throws SQLException	It returns the username of the database.
public String getDatabaseProductName() throws SQLException	It returns the product name of the database.
public String getDatabaseProductVersion() throws SQLException	It returns the product version of the database.
public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types) throws SQLException	It returns the description of the tables of the specified catalog. The table type can be TABLE, VIEW, ALIAS, SYSTEM TABLE, SYNONYM etc.

Table 5: DatabaseMetaData interface methods are listed in.

Example: DatabaseMetaData interface example

```
import java.sql.Connection;;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.SQLException;
public class DBMetaData
```

```
{
        public static void main(String[] args)
        {
                try{
                        Class.forName("com.mysql.jdbc.Driver");
                }
                catch (Exception e)
                {
      System.err.println("Message: " + e.getMessage());
    }
                try{
                        Connection conn=
DriverManager.getConnection("jdbc:mysql://localhost:3306/baou","root","");
                    DatabaseMetaData dbdata = conn.getMetaData();
      System.out.println("Driver Name is: " + dbdata.getDriverName());
      System.out.println("Driver Version is: " + dbdata.getDriverVersion());
      System.out.println("UserName is: " + dbdata.getUserName());
      System.out.println("Database Product Name is: " +
dbdata.getDatabaseProductName());
      System.out.println("Database Product Version is: " +
dbdata.getDatabaseProductVersion());
    }
                catch (SQLException e)
                {
                   e.printStackTrace(System.err);
          System.err.println("SQLState: " + ((SQLException) e).getSQLState());
          System.err.println("Error Code: " + ((SQLException) e).getErrorCode());
          System.err.println("Message: " + e.getMessage());
    }
  }
}
```

After successful compilation it will have a following output:

```
E:\Book\Adv. Java book 2023\Block 2>javac DBMetaData.java

E:\Book\Adv. Java book 2023\Block 2>java DBMetaData
Driver Name is: MySQL-AB JDBC Driver
Driver Version is: mysql-connector-java-5.0.8 ( Revision: ${svn.Revision} )
UserName is: root@localhost
Database Product Name is: MySQL
Database Product Version is: 5.5.39

E:\Book\Adv. Java book 2023\Block 2>_
```

Figure 1: DatabaseMetaData output

ResultSetMetaData object that can be used to get information about the types and properties of the columns in a ResultSet object. Various important methods of ResultSetMetaData interface are detailed in below table 6.

Method	Details
String getColumnName(int column)	It returns the column name of the particular column
String getColumnTypeName(int column)	It returns the datatype of the particular column which we have passed as a parameter
String getTableName(int column)	It returns the table name of the column
String getSchemaName(int column)	It returns the schema name of the column's table
int getColumnCount()	It returns the number of columns of the ResultSet
boolean isAutoIncrement(int Column)	It returns true if the given column is Auto Increment, else false
boolean isCaseSensitive(int Column)	It returns true if the given Column is Case Sensitive, else false

Table 6: ResultSetMetaData interface methods

Example:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
public class RSMetaData {
   private static final String QUERY = "select sno,sname,smob from student";
   public static void main(String[] args) {
```

```
    try{
                    Class.forName("com.mysql.jdbc.Driver");
            }
            catch (Exception e)
            {
        System.err.println("Message: " + e.getMessage());
    }
            try
            {
                    Connection conn=
DriverManager.getConnection("jdbc:mysql://localhost:3306/baou","root","");
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(QUERY);
                    ResultSetMetaData rsmd = rs.getMetaData();

        System.out.println("1. Column count in student table is: " + rsmd.getColumnCount());
        System.out.println("2. First column name in student table is: " +
rsmd.getColumnName(1));
        System.out.println("3. Database name of student table' column id is: " +
rsmd.getCatalogName(1));;
        System.out.println("4. Data type of column id is: " + rsmd.getColumnTypeName(1));
        System.out.println("5. Table name of column id is: " + rsmd.getTableName(1));

    }
            catch (SQLException e)
            {
        e.printStackTrace(System.err);
           System.err.println("SQLState: " + ((SQLException) e).getSQLState());
           System.err.println("Error Code: " + ((SQLException) e).getErrorCode());
           System.err.println("Message: " + e.getMessage());
    }
  }
}
```

After successful compilation it will have a following output:

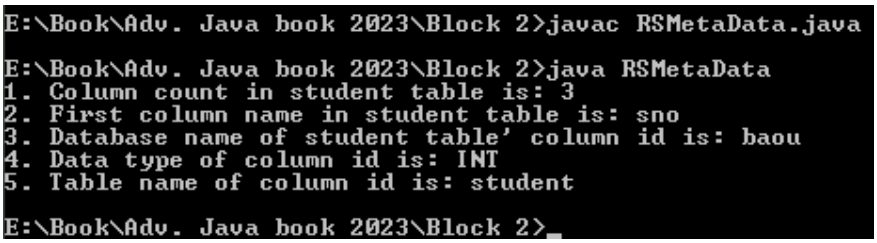


Figure 2: ResultSetMetaData output

Check your progress 4

DatabaseMetaData interface is used to get?????..?

a. Comprehensive information about the database as a whole.

b. Comprehensive information about the table as a whole.

c. Comprehensive information about the column as a whole.

d. Both b and c

2.7 Exceptions

In java.sql package exception can be categorized in following ways:

- SQLException: It is thrown by most methods when there is a problem accessing data.
- SQLWarning: It is thrown to indicate a warning
- DataTruncation: It is thrown to indicate that data may have been truncated
- BatchUpdateException: It is thrown to indicate that not all commands in a batch update executed successfully.

Whenever JDBC encounters an error during an interaction with a data source, it throws an instance of SQLException. The SQLException instance contains the following information that helps to determine the cause of the error:

Description of the error: We can retrieve the String object that contains this description by calling the SQLException.getMessage() method.

SQLState code: These codes and their meanings are standardized by ISO/ANSI and Open Group (X/Open), although some codes are reserved for database vendors to define for themselves. This String object consists of five alphanumeric characters. We can retrieve this code by calling the SQLException.getSQLState() method.

Error code: This is an integer value identifying the error that caused the SQLException instance to be thrown. Its value and meaning are implementation-specific and might be the

actual error code returned by the underlying data source. We can retrieve the error by calling the SQLException.getErrorCode() method.

A cause: A SQLException instance might consists of one or more Throwable objects that caused the SQLException instance to be thrown. To navigate this chain of causes, we need to recursively call the SQLException.getCause() method until a null value is returned.

A reference to any chained exceptions - If more than one error occurs, the exceptions are referenced through this chain. Retrieve these exceptions by calling the method SQLException.getNextException on the exception that was thrown.

Example: Let's consider the previous example code to retrieve the SQLState, error code, error description and cause (if there is one) contained in the SQLException. As we know prevois example was successfully executed. So, to get exception details we need to commit any mistake in code. Here, we are wrongly typing a table column name i.e snno instead of sno in the query as shown below.

```
private static final String QUERY = "select snno, sname, smob from student";
```

After successful compilation, execute the program. It will have a following output:

```
E:\Book\Adv. Java book 2023\Block 2>javac RSMetaData.java

E:\Book\Adv. Java book 2023\Block 2>java RSMetaData
com.mysql.jdbc.exceptions.MySQLSyntaxErrorException: Unknown column 'snno' in 'f
ield list'
        at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:936)
        at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:2985)
        at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:1631)
        at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:1723)
        at com.mysql.jdbc.Connection.execSQL(Connection.java:3277)
        at com.mysql.jdbc.Connection.execSQL(Connection.java:3206)
        at com.mysql.jdbc.Statement.executeQuery(Statement.java:1232)
        at RSMetaData.main(RSMetaData.java:23)
SQLState is: 42S22
Error Code is: 1054
Message is: Unknown column 'snno' in 'field list'
Cause is: null

E:\Book\Adv. Java book 2023\Block 2>
```

Figure 3: Exception details output

SQLWarning objects are a subclass of SQLException that deal with database access warnings. Unlike exceptions, Warnings do not stop the execution of an application; they simply alert the user that something did not happen as expected.

The following subclasses of SQLException can also be thrown:

BatchUpdateException: It will be thrown when an error occurs during a batch update operation. BatchUpdateException provides the update counts for all statements that were executed before the error occurred.

SQLClientInfoException: It will be thrown when one or more client information properties could not be set on a Connection. SQLClientInfoException provides a list of client information properties that were not set.

2.8 Let Us Sum Up

In this unit, we have discussed java.sql package. We learnt the importance of DriverManager class. By using the getConnection method of DriverManager class application connect with the database. Statement helps to execute SQL query, while the results returned by statement will be fetched and manipulated with the help of ResultSet methods. At last we discussed states, codes and other details regarding SQLExceptions.

2.9 Answer for Check Your Progress

Check your progress 1: 1. d 2. b
Check your progress 2: 1. c 2. b
Check your progress 3: 1. d 2. a
Check your progress 4: a

2.10 Glossary

1. Connection: This interface is used to connect a database. The connection object represents communication with database.
2. Statement: The objects created from this interface is used to submit the SQL statements to the database.
3. ResultSet: This object hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
4. SQLException: This class handles any errors that occur in a database application.

2.11 Assignment

1. Discuss various methods of DriverManager class.
2. Write short note on SQLException.
3. What is Statement? Differentiate between execute, executeUpadate() and executeQuery() methods.

2.12 Activities

Write a JDBC application to learn Statement Type and Mode parameter.

2.13 Case Study

Analyse and differentiate important methods of DatabaseMetadata and ResultSetMetadata.

2.14 Further Readings

- Java: The Complete Reference, Eleventh Edition by Herbert Schildt
- https://docs.oracle.com/javase/tutorial/jdbc/overview/index.html
- https://www.javatpoint.com/java-jdbc
- https://www.knowprogram.com/jdbc/jdbc-get-connection/
- https://www.studytonight.com/java/java-sql-package.php

UNIT 3: CONNECTING WITH DATABASE

Unit Structure

3.0 Learning Objectives

After learning this Unit, you will be:

- Able to use different JDBC objects to establish database connection
- Able to write Java programs to connect with databases
- Able to perform DB operations.

3.1 Introduction

Java programs do not store data persistently. Data persistence is generally achieved through relational databases such as Microsoft's SQL Server, the most famous open source database MySQL or NoSQL databases such as MongoDB and Cassandra etc. JDBC provides a uniform API for accessing databases. This API simplifies the development and maintenance of Java programs which interacts with databases. This API includes various classes and interfaces for connecting to a database, running SQL commands and processing the results. Irrespective of the underlying architecture, JDBC makes it possible for the Java programs to communicate with a wide range of databases. In this unit we will discuss various steps involved in database connection.

3.2 Steps to Connect a Database

There are 7 steps involved to connect any java application with the database by using JDBC. They are as follows:

1. Import JDBC packages
2. Load and register the JDBC driver
3. Open a connection to the database
4. Create a statement object
5. Execute the statement and collect a query result
6. Process the result
7. Close the connection

1. Import JDBC Packages

The import statement will ensure that various JDBC API classes will be available to the application program. The following import statement must be included in the program irrespective of the JDBC driver being used:

import java.sql.*;

Import the other classes based on the functionality required in the program.

2. Load and Register the JDBC Driver

Before we proceed to write further code for connecting to the Database, we should load / register the driver in the program. We need to register it only once per database in the program. There are two ways to load the driver in the following ways:

1. Class.forName(): In this technique, the driver's class file is loaded into the memory at runtime. It will implicitly load the driver. While loading, the driver will be registered with JDBC automatically.
2. DriverManager.registerDriver(): This technique will register the driver with the Driver Manager. If the driver is already registered, then it will not take any action.

3. Connecting to a Database

Once the driver is loaded, the next step is to create and establish the connection with database. After the required packages are imported, drivers are loaded and registered, we can proceed for establishing a Database connection. This is achieved by using the getConnection() method of the DriverManager class. The getConnection() requires three input parameters, namely, a connect string (url), a username and a password. In the following code we are creating a connection instance named con.

- Connection con = DriverManager.getConnection(url,user,password)

4. Create a Statement Object

After the connection is established, we will create the statement object to execute the query with the connected database. We will use the createStatement method of the Connection interface to create the query. A call to this method creates an object instance of the Statement interface. The following line of code illustrates this:

- Statement stmt = con.createStatement();

createStatement method is defined in Connection interface and used to execute the sql queries.

5. Executing the Query and collecting a ResultSet

Once a Statement object has been constructed, the next step is to execute the query. This is done by using the executeQuery() method of the Statement object. A call to this method takes a SQL SELECT statement as an argument and returns a JDBC ResultSet object that can be used to get all the records of a table. The following line of code illustrates this using the stmt object created earlier:

- ResultSet rs = stmt.executeQuery ("select empno, empname, empadd, empdept from employee ORDER BY empname");

Where, stmt is the instance of the Statement interface.

6. Processing the results

Once the query has been executed, there are two steps to be carried out:

- Processing the output resultset to fetch the rows
- Retrieving the column values of the current row

To iterate through the data in the ResultSet object, we will call the next() method in a while loop. If there is no more record to read, it will return false.

We can get the data from ResultSet using the getter methods like getXXX() of the JDBC ResultSet object. Here getXXX() corresponds to the getInt(), getString() etc with XXX being replaced by a Java datatype. We have to pass the column index or column name as an argument to get the values using Getter methods.

7. Close the connection

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection. The following code is used to close the connection.

- con.close();

Where, con is the instance of the Connection interface.

Check your progress 1

1. JDBC classes are defined in which packages?

a. jdbc and javax.jdbc

b. java.sql and javax.sql

c. rdb and javax.rdb

d. jdbc and java.jdbc.sql

2. Which of the following method is static and synchronized in JDBC API?

a. getConnection()

b. prepareCall()

c. executeUpdate()

d. executeQuery()

3. Which methods are required to load a database driver in JDBC?

a. registerDriver()

b. forName()

c. getConnection()

d. Both a and b

3.3 JDBC Data Types

As we know that each column in the database table is assigned a SQL type. So, the JDBC driver maps each SQL data type to a JDBC data type. Adapter service for JDBC then maps each JDBC data type to one or more Java data types that are used as the input or output of the adapter service. Generally, data types in JDBC are divided into two categories:

Java data types:

These are the data types supported by the Java programming language. Java data types contain both the primitive (like int, float and double) and the non-primitive data types (like String, Date etc.).

Database data types:

These are the data types supported by the database to which the java application is connected. SQL data types (like int, varchar, date and so on) and NoSQL data types (like BSON, JSON etc.) are the examples of database data types.

While working with JDBC application, we must ensure that the data types in our Java code must match the data types in the database. If there is an int in the java code, we must use an INT in the database. If the Java code contains a String, then we must use a VARCHAR in the database. The below table illustrates some of the most widely used data types.

Java Data Types	**SQL Data Types**
java.lang.string	Varchar
java.math.BigDecimal	Numeric
int	Integer
java.sql.Timestamp	Timestamp
java.sql.Ref	Ref
java.sql.Clob	Clob
java.sql.Time	Date
java.sql.Date	Time
java.sql.Blob	Blob

Table 1: JDBC Data Types

Fortunately, the getXXX() methods of ResultSet perform automatic conversion in many cases. This conversion happens between all numeric types (though you could lose precision in

the conversion) and between most types and String. The Blob type can't be converted automatically to a String.

Check your progress 2

1. What is blob in the following statement?

create table addImage(image blob);

a. Variable

b. Object

c. Data type

d. Class

2. Which data type is used to store files in the database table?

a. CLOB

b. BLOB

c. File

d. Both a and b

3.4 Connectivity with MySQL

After learning the connectivity steps now we will implement all the 7 basic steps to connect with database using JDBC in Java program. We will use MYSQL database for this example. Assume that we have a database named “mca” and table named “student” and sno, sname and smob are the three fields in it.

To connect our Java application with MYSQL we need to have a MySQL Connector JAR file. It is freely available to download. MySQL Connector/J is the official JDBC driver for MySQL. MySQL Connector/J 8.0 is compatible with all MySQL versions starting with MySQL 5.6. Once the MySQL Connector.jar (version may change) file is downloaded it is required to load it for Java application through following ways. There are two ways to load the jar file:

1. First, paste the mysqlconnector.jar file in jre/lib/ext folder and second,

2. Set classpath

There are two ways to set the classpath:

I. Set the temporary classpath by opening a command prompt and write:

- C:>set classpath=c:\folderName\mysql-connector-java-5.0.8-bin.jar;.;

II. Set the permanent classpath. For that go to environment variable then click on new tab. In variable name write classpath and in variable value paste the path as,

- C:\folderName\mysql-connector-java-5.0.8-bin.jar;.;

Now, create a java file and save it as MysqlCon.java.

MysqlConn.java

Load the Driver and connect with database:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.ResultSet;
import java.sql.Statement;
class MysqlConn
{
        Connection con;
        Statement stmt;
        ResultSet rs;
        MysqlConn()
        {
                try
                {
         // Load the Driver
                Class.forName("com.mysql.jdbc.Driver");
            System.out.println("Driver loaded Successfully");
         // Establish the Connection
          con=DriverManager.getConnection("jdbc:mysql://localhost:3306/mca","root","");
                //here mca is database name, root is username and password  is null
                System.out.println("Connection is established Successfully");
                }
                catch (SQLException se)
                {
                        se.printStackTrace();
                }
                catch(Exception e)
```

```
            {
                System.out.println(e);
            }
        }
        public static void main(String args[])
        {
            MysqlConn m=new MysqlConn();
        }
                                        }
```

Now, compile the application as shown below.

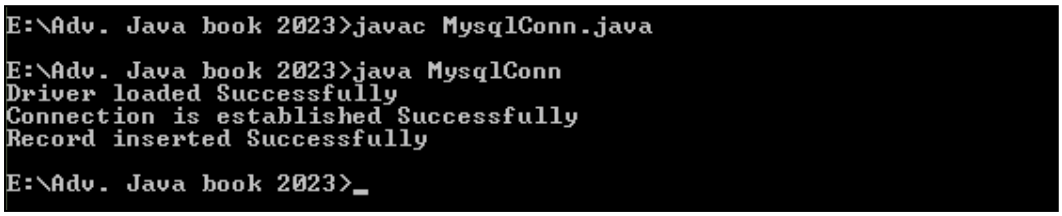
```
E:\Adv. Java book 2023>javac MysqlConn.java

E:\Adv. Java book 2023>java MysqlConn
Driver loaded Successfully
Connection is established Successfully

E:\Adv. Java book 2023>
```

After successful compilation write a code for inserting records in database as shown below.

```
public void insRecord()
        {
            try{
            stmt=con.createStatement();
            stmt.executeUpdate("insert into student values (1001, 'Vinod', 13000)");
            stmt.executeUpdate("insert into student values (1002, 'Dimpal', 14258)");
            stmt.executeUpdate("insert into student values (1003, 'Ved', 15268)");
            stmt.executeUpdate("insert into student values (1004, 'Mukesh', 15600)");
            stmt.executeUpdate("insert into student values (1005, 'Jayram', 18524)");
            stmt.executeUpdate("insert into student values (1006, 'Sujata', 17564)");
            stmt.executeUpdate("insert into student values (1007, 'Aneri', 15268)");
            stmt.executeUpdate("insert into student values (1008, 'Vishal', 18758)");
            stmt.executeUpdate("insert into student values (1009, 'Darshana', 14258)");
            stmt.executeUpdate("insert into student values (10010, 'Nihal', 15268)");
            System.out.println("Record inserted Successfully");

            }
```

```
            catch (SQLException se)
            {
                  se.printStackTrace();
            }
            catch(Exception e)
            {
                  System.out.println("In Insertion:"+e);
            }
      }
      public static void main(String args[])
      {
            MysqlConn m=new MysqlConn();
            m.insRecord();
      }
```

Now, Compile and run the application as shown below.

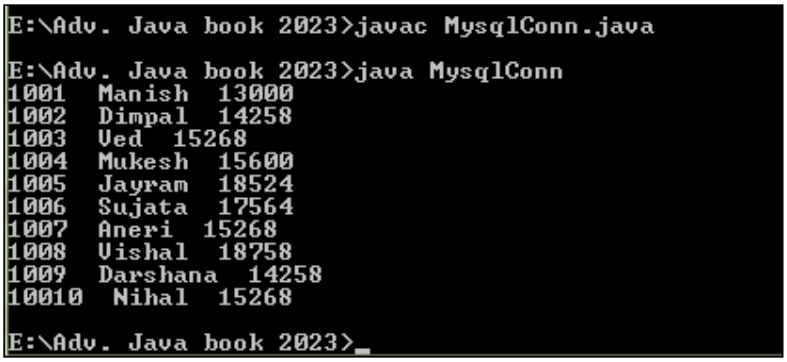


As we can see that above image shows that records are inserted successfully. So, let's query the database to check the records. The required code to select the records is shown below.

```
public void selectRecord()
      {
            try{
            stmt=con.createStatement();
            rs=stmt.executeQuery("select * from student");
            while(rs.next())
                  System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getInt(3));

            }
```

```
            catch (SQLException se)
            {
                  se.printStackTrace();
            }
            catch(Exception e)
            {
                  System.out.println("In selection:"+e);
            }
      }
      public static void main(String args[])
      {
            MysqlConn m=new MysqlConn();
            m.selRecord();
      }
```

After successful compilation run the application and check the output as shown below.



Now, we will try to update the records in the table by name based on students no. The required code for the updation is shown below.

```
public void updRecord(String nam, int p)
      {
            try{
            stmt=con.createStatement();
            stmt.executeUpdate("update student set sname='"+nam+"'where sno="+ p
+"");
            System.out.println("Record Updated Successfully");
```

```
            }
            catch (SQLException se)
            {
                  se.printStackTrace();
            }
            catch(Exception e)
            {
                  System.out.println("In Updation:"+e);
            }
      }
      public static void main(String args[])
      {
            MysqlConn m=new MysqlConn();
            m.updRecord("Manish",1001);
            m.select();
      }
```

Compile and run the application as shown below.

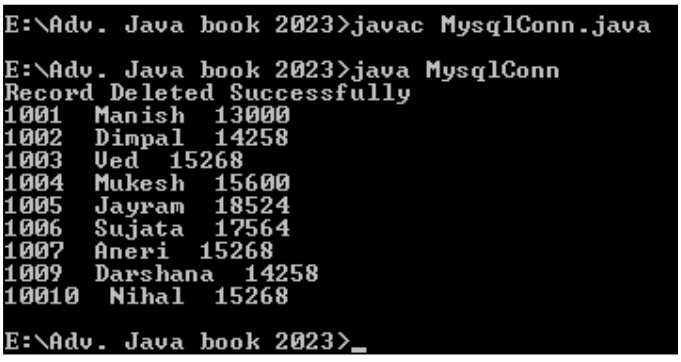
```
E:\Adv. Java book 2023>javac MysqlConn.java

E:\Adv. Java book 2023>java MysqlConn
Record Updated Successfully
1001  Manish  13000
1002  Dimpal  14258
1003  Ved  15268
1004  Mukesh  15600
1005  Jayram  18524
1006  Sujata  17564
1007  Aneri  15268
1008  Vishal  18758
1009  Darshana  14258
10010  Nihal  15268

E:\Adv. Java book 2023>
```

We can see that student no 1001 is successfully updated in the records. Now, we will try to delete the records in the table based on students no. The required code for the deletion is shown below.

```
Public void delRecord(int p)
      {
            try{
            stmt=con.createStatement();
```

```
                stmt.executeUpdate("delete from student where sno="+ p +"");
                System.out.println("Record Deleted Successfully");
                }
                catch (SQLException se)
                {
                        se.printStackTrace();
                }
                catch(Exception e)
                {
                        System.out.println("In Deletion:"+e);
                }
        }
public static void main(String args[])
        {
                MysqlConn m=new MysqlConn();
                m.delRecord(1008);
                m.select();
        }
```

Compile and run the application as shown below.



We can see that student no 1008 is successfully deleted from the records.

Check your progress 3

1. Identify the DSN in the following statement:

DriverManager.getConnection("jdbc:odbc:baou", "ahd", "guj")

a. jdbc

b. odbc

c. baou

d. ahd

2. Which of the following method is used to perform DML statements in JDBC?

a. executeUpdate()

b. executeResult()

c. executeQuery()

d. execute()

3. What should be the correct order to close the database resource?

a. Connection, Statements, ResultSet

b. ResultSet, Statements, Connection

c. ResultSet, Connection, Statements

d. Statements, ResultSet, Connection

3.5 Connectivity with Oracle database

To connect any java application with the oracle database, user needs to follow following steps. In the following example, we are using Oracle 10g as the database. You need to load ojdbc14.jar file for oracle driver. Few things to understand while using oracle database are:

- Driver class: The driver class for the oracle database is oracle.jdbc.driver.OracleDriver.
- Connection URL: The connection URL for the oracle10G database is jdbc:oracle:thin:@localhost:1521:orcl where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we can also use IP address 127.0.0.1, 1521 is the port number and orcl is the Oracle service name. You can get all these information from the tnsnames.ora file.
- Username: The default username for the oracle database is system.
- Password: It is the password given by the user at the time of installing the oracle database.

OracleConn.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.ResultSet;
import java.sql.Statement;
```

```
public class OracleConn
{
	public static void main(String args[])
	{
		try
		{
			//load the driver class
			Class.forName("oracle.jdbc.driver.OracleDriver");
			//create  the connection object
			Connection con=DriverManager.getConnection(
			"jdbc:oracle:thin:@localhost:1521:orcl","system","baou");
			//create the statement object
			Statement stmt=con.createStatement();
			//execute query
			ResultSet rs=stmt.executeQuery("select * from student");
			while(rs.next())
			{
			System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getint(3));
			}
			//close the connection object
			con.close();
		}
		catch (SQLException e)
		{
	System.err.format("SQL State: %s\n%s", e.getSQLState(), e.getMessage());
		 }
		catch (Exception e)
		{
			e.printStackTrace();
		  }
	}
}
```

Check your progress 4

The default username for the oracle database is system?

a. True

b. False

3.6 Let Us Sum Up

In this unit, we have discussed various steps required to develop Java Database application. We have also discussed different data types required to use while developing database application. JDBC is one of Java's oldest APIs which provides an easy - to - use solution for one of the pioneer requirement of Java application development. By just knowing just the few JDBC call discusses in this unit will give you a platform to start using JDBC to connect to virtually any database. Once you have clear understanding of these commands, you can easily start to explore more advanced options that are available into JDBC. Today, as JDBC is enough for simple applications, most developers are eventually looking towards the Jakarta Persistence API to develop a more formal data access layer.

3.7 Answer for Check Your Progress

Check your progress 1: 1. b 2. a 3. d
Check your progress 2: 1. c 2. a
Check your progress 3: 1. c 2. a 3. b
Check your progress 4: a

3.8 Glossary

1. **JDBC Adapter:** The Java Database Connectivity (JDBC) adapter service enables the translation service to communicate with JDBC - compliant databases. The adapter allows for updating or retrieving data from a JDBC - compliant database as part of a business process within the application.

2. BLOB: BLOB is used to hold the binary type of data. The storage size may vary based on the databases. For example: images, voice, video.

3. CLOB: CLOB is used to hold the character type of data. Like BLOB, the storage space may vary based on the DB. For example: files.

4. JAR: JAR file stands for Java ARchive file. Inspired by the popular zip files, Java provides JAR files for accumulating all the components like class files and other

corresponding metadata of all the files which are packaged, including the resources of an application.

5. Thin driver

This driver converts JDBC calls directly into the vendor-specific database protocol. So, it is also known as thin driver. It is fully written in Java language.

3.9 Assignment

1. Discuss various connection steps involved in JDBC applications.
2. Explain the connection string component in detail.
3. Explain various data types of JDBC.
4. Explain various steps of oracle database connectivity.

3.10 Activities

Develop a java database application with Oracle database.

3.11 Case Study

How Java Database Connectivity with MySql differs from MongoDB.

3.12 Further Readings

- Java: The Complete Reference, Eleventh Edition by Herbert Schildt
- https:// docs. oracle. Com / javase / tutorial / jdbc / overview / index. html
- https:// www. javatpoint. Com / java – jdbc
- https:// www. theserverside. com/ definition / Java-Database-Connectivity-JDBC
- https://intellipaat.com / blog / java – jdbc /
- https:// www.ibm.com / docs / en / b2b-integrator / 5.2 ? topic=l-java-database-connectivity-jdbc-adapter-v520-522
- https:// www.scaler.com/ topics / jar-file-in-java/

UNIT 4: WORKING WITH STORED PROCEDURES AND FUNCTIONS

Unit Structure

4.0 Learning Objectives

After learning this Unit, you will be:

- Able to use precompiled statement
- Able to define and use stored procedures
- Able to define transaction
- Able to write batch processing

4.1 Introduction

In the previous unit we discussed and learnt various steps involved in database connectivity. After establishing the database connectivity we performed various operation line Insert, Update and Delete. In this unit we will focus on pre-compiled statements, working with and calling stored procedures to perform various DML operation, transaction management and batch processing.

4.2 Pre-compiled Statement and Stored Procedures

Pre-compiled Statement:

The java.sql.PreparedStatement extends java.sql.Statement and it is used for pre-compiling SQL statements that might contain input parameters. By using java.sql.Statement object in Java application, the SQL query will be fired to database software for multiple times and the same SQL query will be parsed in Database software for multiple times. This will degrade the performance of the application and takes a more time. This problem can be avoided by using pre-compiled SQL query.

Pre-compiled SQL query is the SQL query that goes to database software without input values and becomes parsed or compiled SQL query in database software irrespective of whether it will be executed or not. PreparedStatement object represents this pre-compiled SQL query. We can use this object to set input values to SQL query, to execute SQL query and to fetch the output of SQL query for one or multiple times. PreparedStatement object is excellent feature to execute the same SQL query for multiple times either with or without input values.

The PreparedStatement interface extends the Statement interface. It represents pre-compiled SQL statements and stores it in a PreparedStatement instance. It improves the performance of the application because the query is compiled only once. In the following code we can see a parameterized query:

```
String sql=”insert into student values(?,?,?)”;
```

Here, the parameter (?) is passed for values which will be set by calling setter methods of PreparedStatement interface. The important methods of PreparedStatement interface are listed in table 1.

Method	Details
public void setInt(int paramIndex, int value)	It sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	It sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	It sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	It sets the double value to the given parameter index.
public int executeUpdate()	It executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	It executes the select query. It returns an instance of ResultSet.

Table 1: Methods of PreparedStatement Interface

Example: The following example demonstrates the use of PreparedStatement Interface for Insert, Update and Delete operation. We have a created three column in database named sno,sname and smob for CRUD operation.

Note: All the parameter / arguments are represented by "?" symbol and each parameter is referred to by its original position.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.PreparedStatement;
class MysqlConnPreStmt
{
        Connection con;
        Statement stmt;
        ResultSet rs;
```

```
        MysqlConnPreStmt()
        {
                try
                {
                        Class.forName("com.mysql.jdbc.Driver");
                        System.out.println("Driver loaded Successfully");

        con=DriverManager.getConnection("jdbc:mysql://localhost:3306/baou","root","");
                        //here baou is database name, root is username and password  is null
                        System.out.println("Connection is established Successfully");
                }
                catch (SQLException se)
                {
                        se.printStackTrace();
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
        }
        public void select()
        {
                try{
                stmt=con.createStatement();
                rs=stmt.executeQuery("select * from student");
                while(rs.next())
                        System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getInt(3));
                }
                catch (SQLException se)
                {
                        se.printStackTrace();
                }
                catch(Exception e)
                {
                        System.out.println("In selection:"+e);
                }
        }
        public void ins()
        {
                try{
                    PreparedStatement ps = con.prepareStatement("insert into student values(?,
?, ?)");
          ps.setInt(1, 2001);
          ps.setString(2, "Manish");
          ps.setInt(3, 98989);
                        ps.executeUpdate();
          System.out.println("Record inserted Successfully");

                }
                catch (SQLException se)
```

```
                {
                        se.printStackTrace();
                }
                catch(Exception e)
                {
                        System.out.println("In Insertion:"+e);
                }
        }
        public void del(int p)
        {
                try{
                        PreparedStatement stmt=con.prepareStatement("delete from student
where sno=?");
                        stmt.setInt(1,p);
                        int i=stmt.executeUpdate();
        System.out.println(i+" records deleted");
                }
                catch (SQLException se)
                {
                        se.printStackTrace();
                }
                catch(Exception e)
                {
                        System.out.println("In Insertion:"+e);
                }
        }
        public void upd(String nam,int p)
        {
                try{

                        PreparedStatement stmt=con.prepareStatement("update student set
sname=? where sno=?");
                stmt.setString(1,nam);  //1 specifies the first parameter in the query i.e. name
                        stmt.setInt(2,p);
                        int i=stmt.executeUpdate();
        System.out.println(i + " record updated");
                }
                catch (SQLException se)
                {
                        se.printStackTrace();
                }
                catch(Exception e)
                {
                        System.out.println("In Updation:"+e);
                }
        }
        public static void main(String args[])
        {
                MysqlConnPreStmt m=new MysqlConnPreStmt();
                m.select();
```

```
                m.ins();
                m.select();
                m.upd("Naresh",2001);
                m.select();
                m.del(2001);
                m.select();

        }
}
```

After successful compilation and execution, it will display following output on the prompt.

Figure 1: PreparedStatement Example

Limitation of PreparedStatement

- Using one simple statement object we can execute different types of SQL queries but one after another.
- We can execute only one type of query using one PreparedStatement object for which it is confined. At a time one PreparedStatement will have only one compiled query. To execute multiple different types of queries one needs to take multiple prepared statement objects.

CallableStatement:

Stored procedures are created with the help of a group of SQL statements that perform a particular task. These procedures accept input and may return a value. Stored procedures are useful when we want to reuse the command multiple times. The CallableStatement interface is used to execute SQL stored procedures. The JDBC API provides a stored procedure SQL escape syntax supports stored procedures to be called in a standard way for all RDBMSs. The prepareCall(String SQL) method of the Connection interface creates a CallableStatement object for calling database stored procedures. The CallableStatement object have various methods for setting up its IN and OUT parameters along with methods for executing the call to a stored procedure.

• We can invoke one of the following methods to call the stored procedure:

- If the stored procedure does not return result sets use CallableStatement.executeUpdate method.
- If the stored procedure returns one result set use CallableStatement.executeQuery method.
- If the stored procedure returns multiple result sets use CallableStatement.execute method.

We can use the CallableStatement.getXXX methods to retrieve values from the OUT parameters or INOUT parameters. When this object is no more required then CallableStatement.close method is called to close the CallableStatement object.

Example:

To understand the example we have created four stored procedures in our database named Display, inStud, Modify and delet as shown in below figure 2.

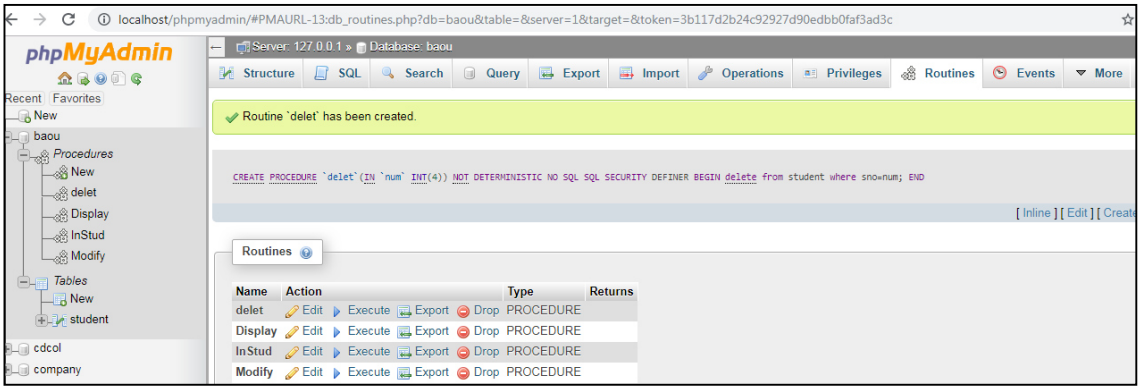


Figure 2: MYSQL Stored Procedure

Steps to work with stored procedure in JDBC MySQL:

- First, open a connection to MySQL server by creating a new Connection object.
  Connection conn = DriverManager.getConnection();
- Second, prepare a stored procedure call and create a CallableStatement object by calling prepareCall() method of the Connection object.
  String str = "{CALL inStud(?,?,?)}";
  CallableStatement stmt = conn.prepareCall(str)
- Third, pass all the parameters to the stored procedure. Here, the inStud stored procedure accepts only three IN parameter.

- At last, execute the stored procedure by calling the executeQuery() or executeUpdate() method of the CallableStatement object.

```
// MysqlConnCallStmt.java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.CallableStatement;
class MysqlConnCallStmt
{
	Connection con;
	Statement st;
	CallableStatement stmt = null;
	ResultSet rs;
	MysqlConnCallStmt()
	{
		try
		{
			Class.forName("com.mysql.jdbc.Driver");
			System.out.println("Driver loaded Successfully");

	con=DriverManager.getConnection("jdbc:mysql://localhost:3306/baou","root","");
			//here baou is database name, root is username and password  is null
			System.out.println("Connection is established Successfully");
		}
		catch (SQLException se)
		{
			se.printStackTrace();
		}
		catch(Exception e)
		{
```

```
                System.out.println(e);
        }
}
public void select()
{
        try{

                String str="{call Display()}";
            stmt = con.prepareCall(str);
 rs=stmt.executeQuery();
        while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getInt(3));
        }
        catch (SQLException se)
        {
                se.printStackTrace();
        }
        catch(Exception e)
        {
                System.out.println("In selection:"+e);
        }
}
public void ins()
{
        try{
                  String str="{call inStud(?,?,?)}";
            stmt = con.prepareCall(str);
 stmt.setInt(1, 2004);
 stmt.setString(2, "Kalpesh");
 stmt.setInt(3, 45663);
                stmt.executeUpdate();
 System.out.println("Record inserted Successfully");
```

```
            }
            catch (SQLException se)
            {
                  se.printStackTrace();
            }
            catch(Exception e)
            {
                  System.out.println("In Insertion:"+e);
            }
      }
      public void del(int p)
      {
            try{
                  String str="{call delet(?)}";
                stmt = con.prepareCall(str);
       stmt.setInt(1, p);
       stmt.executeUpdate();
        System.out.println("Record deleted Successfully");
            }
            catch (SQLException se)
            {
                  se.printStackTrace();
            }
            catch(Exception e)
            {
                  System.out.println("In Insertion:"+e);
            }
      }
      public void upd()
      {
            try{

                  String str="{call Modify(?,?)}";
```

```
                    stmt = con.prepareCall(str);
          stmt.setInt(1, 2002);
          stmt.setString(2, "Mukesh");
          stmt.executeUpdate();
          System.out.println("Record Updated Successfully");


                }
                catch (SQLException se)
                {
                        se.printStackTrace();
                }
                catch(Exception e)
                {
                        System.out.println("In Updation:"+e);
                }
        }
        public static void main(String args[])
        {
                MysqlConnCallStmt m=new MysqlConnCallStmt();
                m.select();
                m.ins();
                m.select();
                m.upd();
                m.select();
                m.del(2001);
                m.select();


        }
}
```

Note: The MysqlConnCallStmt.java file inserts, updates and deltes the record in student table in MySQL database using the stored procedure as shown in figure 3.

```
E:\Book\Adv. Java book 2023\Block 2>javac MysqlConnCallStmt.java

E:\Book\Adv. Java book 2023\Block 2>java MysqlConnCallStmt
Driver loaded Successfully
Connection is established Successfully
2001  Naresh  25254
2002  Kaushik  89754
2003  Mitesh  65412
Record inserted Successfully
2001  Naresh  25254
2002  Kaushik  89754
2003  Mitesh  65412
2004  Kalpesh  45663
Record Updated Successfully
2001  Naresh  25254
2002  Mukesh  89754
2003  Mitesh  65412
2004  Kalpesh  45663
Record deleted Successfully
2002  Mukesh  89754
2003  Mitesh  65412
2004  Kalpesh  45663

E:\Book\Adv. Java book 2023\Block 2>
```

Figure 3: MYSQL Stored Procedure output

Check your progress 1

1. Which of the following is efficient than a statement due to the pre-compilation of SQL?

a. PreparedStatement

b. Statement

c. CallableStatement

d. None of these

2. Parameterized queries can be executed by?

a. ParameterizedStatement

b. PreparedStatement

c. CallableStatement and Parameterized Statement

d. All of these

3. Which interface facilitates to store images in the database?

a. PreparedStatement

b. ResultSetMetaData

c. DatabaseMetData

d. None of these

4.3 Transaction Management

A transaction is a group of SQL statements that need to be either executed all successfully or not at all. Failure to perform even one statement leads to an inconsistent and erroneous database. Any database must satisfy the ACID properties (Atomicity, Consistency, Isolation, and Durability) to guarantee regarding the successful execution of a database transaction.

1. Atomicity: It states that each transaction should be carried out as a whole part; if one part of the transaction fails, then the whole transaction fails.
2. Consistency: It states that the database should be in a valid state before and after the transaction is performed.
3. Isolation: It describes that each transaction must execute in complete isolation without knowing the existence of other transactions.
4. Durability: It states that once the transaction is complete, the changes made by the transaction are permanent.

Disabling Auto-Commit Mode

Initially, when a connection is created it is in auto-commit mode. This means that each individual SQL statement is treated as a separate transaction and is automatically committed after it is executed. The mechanism to allow two or more statements to be grouped into a single transaction is to disable the auto-commit mode. This is shown in the following code, where the conn is an active connection:

- conn.setAutoCommit(false);

Committing Transactions

After the auto-commit mode is disabled, no SQL statements are committed until the commit method called explicitly. All statements executed after the previous call to the commit method are included in the current transaction and committed together as a group. In JDBC API, the Connection interface provides the setAutoCommit() , commit() and rollback() methods to perform transaction management operation

The following steps are required to perform for transaction management in JDBC API.

- Disable auto-commit mode by passing the false value to the setAutoCommit() method.
- Call the commit() method to commit the transaction if all statements are executed successfully.
- Call the rollback() method to cancel the transaction if any one of statements fails.

Following example demonstrates the above steps.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.PreparedStatement;
class JDBCTransaction
{  private static Connection conn;
        public static void main(String args[])
        {

                try{
                        Class.forName("com.mysql.jdbc.Driver");
                }
                catch (Exception e)
                {
        System.err.println("Message: " + e.getMessage());
      }
                try
                {
                        conn=
DriverManager.getConnection("jdbc:mysql://localhost:3306/baou","root","");
                        // Disable auto commit mode
        conn.setAutoCommit(false);
                        // Insert
                        PreparedStatement ps = conn.prepareStatement("insert into studen
values(?, ?, ?)");
          ps.setInt(1, 2001);
          ps.setString(2, "Manish");
          ps.setInt(3, 98989);
                    ps.executeUpdate();
          System.out.println("Record inserted Successfully");
```

```
                        // Update
                        PreparedStatement ps1 = conn.prepareStatement("update student se
sname = ? where sno = ?");
          ps1.setInt(2, 2001);
          ps1.setString(1, "Mukesh");
          ps1.executeUpdate();
          System.out.println("Record Updated Successfully");
                                // Commit insert and update statement
           conn.commit();
           System.out.println("Transaction is commited successfully.");
                }
                catch (SQLException e)
                {
                        e.printStackTrace(System.err);
        System.err.println("SQLState is: " + ((SQLException) e).getSQLState());
        System.err.println("Error Code is: " + ((SQLException) e).getErrorCode());
        System.err.println("Message is: " + e.getMessage());
                        System.err.println("Cause is: " + e.getCause());
                        if (conn != null)
                        {
           try {
              // Roll back transaction
              System.out.println("Transaction is being rolled back.");
              conn.rollback();
              }
                                        catch (Exception ex)
                                        {
               ex.printStackTrace();
              }
                        }
     }
        }
}
```

Output:

```
E:\Book\Adv. Java book 2023\Block 2>javac JDBCTransaction.java

E:\Book\Adv. Java book 2023\Block 2>java JDBCTransaction
Record inserted Successfully
Record Updated Successfully
Transaction is commited successfully.

E:\Book\Adv. Java book 2023\Block 2>javac JDBCTransaction.java
```

Figure 4: Commit output

Now, suppose by mistake we makes a typing mistake in writing a "snamee" instead of "sname" in update statement. So, after compilation when we execute the program it will display output as shown in figure 5.

```
E:\Book\Adv. Java book 2023\Block 2>javac JDBCTransaction.java

E:\Book\Adv. Java book 2023\Block 2>java JDBCTransaction
Record inserted Successfully
com.mysql.jdbc.exceptions.MySQLSyntaxErrorException: Unknown column 'snamee' in
'field list'
        at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:936)
        at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:2985)
        at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:1631)
        at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:1723)
        at com.mysql.jdbc.Connection.execSQL(Connection.java:3283)
        at com.mysql.jdbc.PreparedStatement.executeInternal(PreparedStatement.ja
va:1332)
        at com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java
:1604)
        at com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java
:1519)
        at com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java
:1504)
        at JDBCTransaction.main(JDBCTransaction.java:36)
SQLState is: 42S22
Error Code is: 1054
Message is: Unknown column 'snamee' in 'field list'
Cause is: null
Transaction is being rolled back.
```

Figure 5: Rolled Back output

Check your progress 2

1. Transaction-related methods are supported in the Connection interface.

a. True

b. False

2. You can use setAutoCommit(false) to enable manual commits.

a. True

b. False

3. In Transaction Management the JDBC Transaction represents?

a. Single unit of work

b. Multiple unit of work

c. Both a & b

d. None of these

4. The ACID properties does not describes the transaction management well.

a. True

b. False

4.4 Batch Processing

Sometimes we need to run bulk queries of a similar type to a database. For example, loading a bulk data from CSV files to the relational database table. As we know that Statement or PreparedStatement are the options with us to execute queries. Apart from this JDBC provides Batch Processing functionality through which we can execute the bulk of queries in one go for a database. We can batch both SQL inserts, updates and deletes. It does not make any sense to batch select statements. The Statement interface provides two methods to perform batch operations:

- addBatch(String sql)
- executeBatch()

Statement.addBatch(String sql)

This method will add the given SQL command to the current list of commands for this Statement object. The commands in this list will be executed as a batch by calling the method executeBatch.

Statement.executeBatch()

This method will submit a batch of commands to the database for execution and if all commands execute successfully it will return an array of update counts.

In the following example we have discussed batch processing for insert statement for the Student database.

```
import java.sql.BatchUpdateException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Arrays;

class BatchprocExample
{
        private static Connection conn;
```

```
        public static void main(String args[])
        {

                try{
                        Class.forName("com.mysql.jdbc.Driver");
                }
                catch (Exception e)
                {
        System.err.println("Message: " + e.getMessage());
      }
                try
                {
                        conn=
DriverManager.getConnection("jdbc:mysql://localhost:3306/baou","root","");
                        Statement stmt = conn.createStatement();
        conn.setAutoCommit(false);
                         stmt.addBatch("insert into student values(1,'Vraj',1254);");
                         stmt.addBatch("insert into student values(2,'Nirav',4587);");
                         stmt.addBatch("insert into student values(3,'Shrey',3698);");
                         stmt.addBatch("insert into student values(4,'Himanshu',4568);");

                         int[] updateCounts = stmt.executeBatch();
         System.out.println(Arrays.toString(updateCounts));
         conn.commit();
          System.out.println("Records inserted Successfully");

                }
                catch (BatchUpdateException b)
                {
                        System.err.println("BatchUpdateException");
                       System.err.println("SQLState is:  " + b.getSQLState());
                      System.err.println("Message is:  " + b.getMessage());
                      System.err.println("Vendor is:  " + b.getErrorCode());
```

```
                    System.err.print("Update counts is:  ");
        }
                catch (SQLException e)
                {
        e.printStackTrace(System.err);
        System.err.println("SQLState is: " + ((SQLException) e).getSQLState());
        System.err.println("Error Code is: " + ((SQLException) e).getErrorCode());
        System.err.println("Message is: " + e.getMessage());
        System.err.println("Cause is: " + e.getCause());
                }
        }
}
```

Output:

```
E:\Book\Adv. Java book 2023\Block 2>javac BatchprocExample.java

E:\Book\Adv. Java book 2023\Block 2>java BatchprocExample
[1, 1, 1, 1]
Records inserted Successfully

E:\Book\Adv. Java book 2023\Block 2>
```

Figure 6: Batch Processing output

Check your progress 3

1. Which of the following interface provides the commit() and rollback() methods?

a. Statement Interface

b. ResultSet Interface

c. Connection Interface

d. RowSet Interface

2. Which interfaces provide methods for batch processing in JDBC?

a. java.sql.Statement

b. java.sql.PreparedStatement

c. Both a & b

d. None of these

4.5 Let Us Sum Up

In this unit, we have discussed the importance and situation for the implementation and use of Prepared Statement and callable statement. We also discussed the concept of transaction management with reference to commit and rollback statement. In a transaction, if one statement fails then all statements of the transaction gets cancelled or rolledback. If all statements in a transaction are successfully executed, then only the transaction status changes to successful means commit. A transaction always follows the principle of All or Nothing. At last, we covered batch processing. It refers to running batch (set) jobs on a computer system. Batch Processing allows to group related SQL statements into a group and submit them with one call to the database system.

4.6 Answer for Check Your Progress

Check your progress 1: 1. a 2. b 3. a
Check your progress 2: 1. a 2. a 3. a 4. b
Check your progress 3: 1. c 2. c

4.7 Glossary

1. Prepared Statement: This statement is used to execute parameterized query.

2. Commit: This statement in SQL ends a transaction within a relational database management system (RDBMS) and saves all changes and makes it visible to other users.

3. ROLLBACK: This command causes all data changes to be discarded in the relational database management. It restores a database to a previous state by cancelling a specific transaction or set of transaction.

4.8 Assignment

1. Differentiate between Statement and PreparedStatement.
2. Explain the importance of CallableStatement with example.
3. Write a short note on commit and rollback with reference to transaction management.

4.9 Activities

Analyse and differentiate important methods of PreparedStatements and CallableStatement.

4.10 Case Study

Analyse the various situations for the implementation of Batch Processing.

4.11 Further Readings

- https://www.tutorialspoint.com/jdbc/jdbc-batch-processing.htm
- https://docs.oracle.com/javase/tutorial/jdbc/overview/index.html
- https://javaee.github.io/tutorial/batch-processing001.html
- https://ecomputernotes.com/java/jdbc/jdbc-transaction

BLOCK 3: Web Application, Servlets and Session Management

Block Introduction

In this block, we will discuss in detail about the fundamental concept of Java web application i.e. Servet along with its architecture and API. The student will be demonstrated practically on how to configure and run Servlet in Apache Tomcat 7.

The block focuses on client server environment, how they interact to communicate, HTTP protocol to be used. Every web application has its own implementation structure based on underlying language and server. Here, we have also discussed the web application project structure to be deployed in apache tomcat server. The student will also learn and understand about the basics of configuring various parameters in web.xml file (deployment descriptor file) for Servlets. The block also focuses on HTTP methods, request redirection and various types of session handling mechanism to store the state of the application.

Block Objective

After learning this block, you will be able to:

- Define the concept and Architecture of the Servlet
- Configure, Compile and run Servlets in Apache Tomcat 7
- Configure deployment descriptor file
- Differentiate various HTTP methods
- Differentiate Servlet Config and Servlet Context
- Redirect the request to the specific servlet
- Maintain and work with Session handling mechanism

Block Structure

Unit 1:	Basics of a Web Application
Unit 2:	Servlets
Unit 3:	Servlet Collaboration and Configuration
Unit 4:	Session Management

Block Summary

In this block the Servlet API was discussed and focus was placed specifically on Servlets. Servlets are the fundamental building block of server-side Java programming. A Servlet is highly scalable and easily outperforms traditional CGI with its simple life cycle i.e. initialization, service and destruction. Commonly, the term Servlets actually refers to HTTP Servlets used on the World Wide Web. Various HTTP methods are also discussed with specific emphasis on GET and POST. The HttpServlet class is designed especially for simplifying server-side Java support for HTTP. The students have been well explained on compiling, configuring and executing Servlets in Apache Tomcat 7 practically. Students have also been well explored by different Session tracking mechanisms and their applications.

Block Assignment

Short Answer Questions:

1. Define Servlet and its life cycle.
2. Explain the Threading Issues in servlets.
3. Define servlet context.
4. Explain the functionality of request dispatcher.
5. Define Cookie.
6. Define HttpSession.

Long Answer Questions:

1. Discuss in detail about Architecture of the Servlet.
2. Explain the web application directory structure of Apache Tomcat 7.
3. Write a short note on configuring web.xml file for servlets.
4. Differentiate Http Get and Http Post method.
5. Write a program to explain state management using Http Session.

UNIT 1: Basics of a Web application

Unit Structure

1.0 Learning Objectives

After learning this unit, you will be:

- Able to define web application
- Able to discuss web client and web server
- Able to understand various protocols used in web application
- Able to understand tomcat server as container

1.1 Introduction

A web application is known as an application program which will be stored on a remote server and delivered over the internet through a browser interface. Programmers develop web applications for a various purpose and for different users including an organization or an individual. Today we can finds various web applications on the internet including public service application, online tax calculation or e-commerce applications. In this unit, you will learn the basics of web application, various protocols used in web applications and the web container which contains web applications.

1.2 Web Application

The World Wide Web (WWW or the Web) is an information space where all kinds of documents and other web resources are available and accessed by Uniform Resource Locators. These documents are interlinked by hypertext links and can be accessed through Internet. World Wide Web was invented by english scientist Tim Berners-Lee in 1989. The World Wide Web has been a heart core for the development of the information and is the basic tool that people using to interact over the Internet. Web pages are basically a text documents developed with the help of Hypertext Markup Language (HTML). These documents contain formatted text, images, video, audio and other software components that are rendered in the web browser as a logical group of multimedia content. Embedded hyperlinks allow users to navigate between different web pages. Website is consisting of multiple web pages with a common theme, a common domain name or both. The content on the website is generally provided by the publisher or users contribute the content through their interaction with the website. Websites can be categorized as informative, entertainment, commercial, governmental or non-governmental organizational purposes.

A Uniform Resource Locator (URL) is the address of a document found on the WWW. Browser fetches the URL and interprets the information in the URL in order to connect to the

proper Internet server and to retrieve the desired document. Whenever a user clicks on a hyperlink in a WWW document, it instructs browser to find the URL that is embedded within the hyperlink.

The components in a URL are:

- Protocol://server's address/filename

The protocol may be Hypertext Protocol (http), File Transfer Protocol (ftp), Telnet Protocol etc.

Another terminology with reference to Web Application access is Domain. Domains divide World Wide Web sites into categories depending on the nature of their owner and they form part of a website's address or uniform resource locator (URL).

In the internet address: https://www.google.com, the ".com" part is known as TLD (Top-Level Domain). TLDs are basically classified into two categories: generic TLDs and country-specific TLDs. Some examples of top-level domains are:

.com: commercial enterprises

.mil: military site

.org: organization site (non-profits, etc.)

.int: organizations established by international treaty .net-network

.biz: commercial and personal

.edu: educational site (universities, schools etc.)

.info: commercial and personal

.gov: government organizations

.us: United States

.ca: Canada

.uk: United Kingdom

.in: India

.au: Australia

Some common advantages of web applications include the following:

- Multiple users can access the same version of the application
- Users don't need to install the application at client side
- Cross platform compatibility can be achieved as users can access the application through various platforms such as a desktop, laptop or mobile.
- Users can access the application through multiple browsers like Internet Explorer, Firefox, Chrome and Safari.

Check your progress 1

1. WWW stands for _____.

a. Web World Wide

b. World Wide Web

c. World Web Wide

d. None of these

2. WWW is also known as _____.

a. W2

b. W4

c. W3

d. W1

3. _____ is a network of interconnected hypertext documents that may be accessed via the Internet.

a. World Wide Web

b. W3

c. Both a) and b)

d. None of these

4. A _____ is a software program that is used to browse Web sites and serves as a bridge between the user and the World Wide Web.

a. World Wide Web

b. Web Server

c. Web Browser

d. None of these

1.3 Web Client and Web Server

A web client is software program installed on the user's device that accesses an internet to send an HTTP request and processing the resulting HTTP response. A web client is a software application that communicates to a web server using Hypertext Transfer Protocol (HTTP). On the internet browser is the most commonly used web client. The most common and widely used interface to the World Wide Web is a browser such as Chrome, Mozilla Firefox, Netscape Navigator or Internet Explorer. The basic function of a web browser is to render HTML, the language used to design or mark up webpages. Each time a browser loads a web page, it processes the HTML containing text, links, references to images and other

items like cascading style sheets and JavaScript functions. The browser processes all these items and then renders them on the browser window.

All web browsers provide some standard features which aim to make internet browsing easier for users. Generally, the web client should have some of the following features:

- Support for private or incognito browsing
- Support for VPN or Proxy support
- Support for Multiple tabs or windows
- Support for Back or Previous and Forward buttons
- Home, Refresh, Stop buttons
- Address bar for URL
- Maintain history and bookmark

As we all know that web clients cannot do much on their own. The data they display on browsers screen is driven by web servers. So, Web servers are the computer that supplies files or services to the requesting computer over the internet. So, web servers are systems must be connected to the internet to serve the request and it store web pages. In addition to this it sends requested data over the internet using HTTP. Simply, web servers are just like collection of libraries for web pages responsible for storing, processing and delivering the web resources to the client software.

The web server is responsible for storing files or documents and processing them. Today, all modern web servers possess the following common features:

- Support standard protocols like HTTP, FTP, SMTP and SSH
- Can deliver static content as requested and dynamic content on demand
- Maintain logging data of client and requests
- Support virtual hosting
- Ability to authorize or deny website path traversal
- It must have support for large files and custom error pages

There are various types of server software available for delivering web content to end users. Some of the popular web servers include Apache, Microsoft IIS (Internet Information Services) and Nginx.

Web Servers and Web Clients Communication:

Whenever a user searches for a website, he / she type its URL in the browser. The browser then translates the web address to its IP address. An IP address is a number that uniquely identifies devices on the internet. This information is available and stored on DNS name

servers. The browser queries the DNS database for the IP address of users URL and retrieves it. Once the IP address is traced or found, user's web client sends an HTTP request to the server's IP address. Then the server will receive the request and sends an HTTP "200 OK" message if the requested data is available. Then it sends back the requested data in small chunks known as packets.

Check your progress 2

1. The computer provides services called _____.

a. Client

b. Server

c. Node

d. None of these

2. The computer uses the services which are provided from the server is called _____.

a. Client

b. Server

c. Node

d. None of these

1.4 HTTP Protocol

HTTP means HyperText Transfer Protocol. HTTP is the underlying protocol used by the World Wide Web and it defines how messages will be formatted and transmitted. It is an application layer protocol. It is mainly used for the retrieval of information from websites throughout the internet. It works on the top of TCP/IP protocol. HTTP uses a client-server model where Web browser is the web client and it communicates with the web server which is hosting the website. The request and response messages will carry data in the form of a MIME like format.

The main features of the HTTP are:

- It is a connectionless protocol,
- It is media independent protocol,
- It is stateless protocol

Http Protocol uses two request-response methods between client and server such as GET and POST to handle form submissions. GET Method requests data from a specified resource using

a given URI and POST Method request submits data to process to a specified resource to the web server.

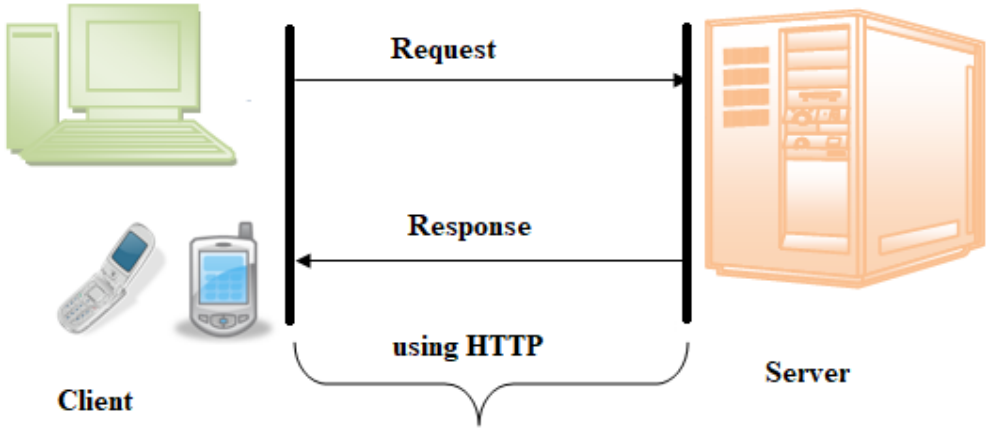


Figure 1: Client Server Communication through HTTP

Whenever a client makes a request for some information (or clicks on the hyperlink) to the web server, the browser sends a request message to the HTTP server for the requested data. After that the following process will takes place:

- The Connection will be opened between the client and the web server through the TCP.
- Then after, the HTTP sends a request to the web server that mainly collects the requested data.
- The response with the data is sent back to the client by HTTP
- At the end, HTTP closes the connection.

HTTPS: HTTPS stands for Hyper Text Transfer Protocol Secure. Basically, it is the secure version of HTTP. Communications between the browser and website are encrypted by Transport Layer Security (TLS) or its predecessor, Secure Sockets Layer (SSL).

Check your progress 3

1. On which port number does HTTP protocol works?

a. 23

b. 21

c. 25

d. 80

2. HTTP is a Stateless Protocol.

a. True

b. False

1.5 Web Container

In Java, a web container is the part of a web server that interacts with Java servlets. A web container manages the life cycle of servlets object; it maps a URL to a particular servlet while ensuring that the requester has relevant access-rights as per the deployment descriptor (web.xml) file.

The web container implements the web component area of the java engineering structure. It specifies a run time environment for various components like security, concurrency, transaction and deployment.

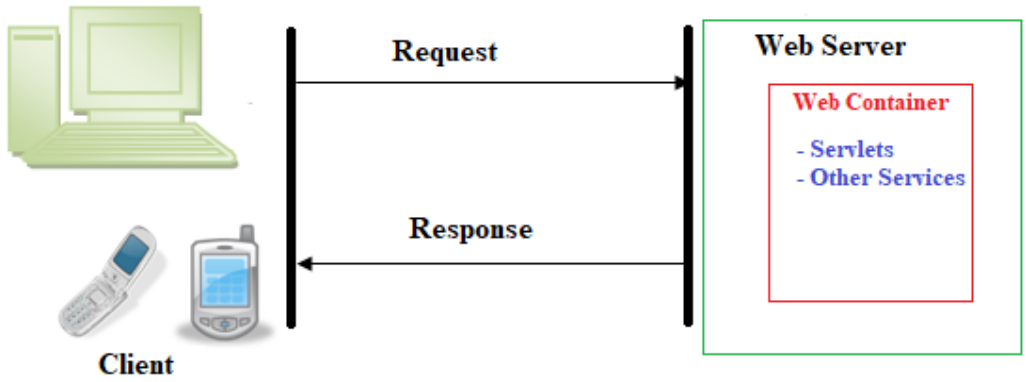


Figure 2: Web Container

There are a various Servlet Containers like Jboss, Apache Tomcat, WebLogic. Apache Tomcat is a open source Java servlet container that implements core java enterprise (now Jakarta EE) specifications, which includs the Jakarta Servlet, Jakarta Server Pages, and Jakarta WebSocket specifications. Initially, tomcat started as the reference implementation for the original Java Servlet API and JavaServer Pages specification. Now, it remains the most widely accepted Java application server, a well-tested and proved core engine with good extensibility. You can download the latest version of Tomcat from the following link:

- https://tomcat.apache.org/

After successfully installing and initialising Tomcat on your local machine, you can verify whether the tomcat is running or not by entering the URL: http://localhost:8080.

The most significant benefit of using Tomcat are as follows:

- It is open-source
- It is incredibly lightweight
- It is highly flexible
- Today, it is one of the most stable platforms to build and run our applications.

- It provides us an extra level of security
- It is well documented
- It is one of the most widely used application servers

Check your progress 4

1. Tomcat is an application server.

a. True

b. False

2. The default port for Tomcat is _____.

a. 8079

b. 8080

c. 8081

d.8088

3. Which of the following is not a Web Server.

a. Tornado

b. Jetty

c. Tomcat

d. BlueGriffon

1.6 Web application Project Structure

The typical directory structure of a Tomcat installation consists of the following:

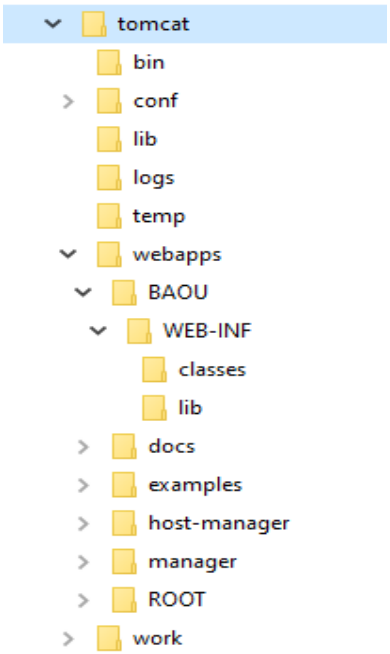


Figure 3: Tomcat Directory structure

bin: This directory contains startup, shutdown and other scripts and executables

lib: This directory contains required library in the form of jar files

conf: This directory contains XML files and related DTDs to configure Tomcat

logs: This directory contains Catalina and application logs

webapps: This directory contains the web applications

work: This directory contains temporary storage for files and directories

examples: This directory contains demo examples of servlet and jsp

common: This directory contains common classes that Catalina and web applications can use

In figure 3, we can see that BAOU is the Web application developed under the webapp directory. So, all the web application goes inside the webapp directory. All the static files will be stored in webapp directory but outside the WEB-INF directory. Tomcat hides the contents of this directory from users for security reasons. This directory is the location where all Java class files are stored in classes directory along with the web.xml file. This web.xml file defines a number of parameters for the application including security information and the mapping of user requests like URIs to servlets. Apart from classes and web.xml, this directory contains another important directory called lib; which will have various jar files required by tomcat to run the application.

Check your progress 5

1. The HTTP request line contains a _____ method to request a document from the server.

a. GET

b. POST

c. COPY

d. None of these

2. What are the two types of connectors used in tomcat?

a. Http Connector and AJP connector

b. Http Connector and JSP connector

c. JSP Connector and AJP connector

d. None of these

1.7 Let Us Sum Up

In this unit we have learnt web application, its container and the role of WWW as a pillar of client - server communication. The program user use to access the WWW is known as a browser because it browses the WWW and requests these hypertext documents to the web server. We also discussed the uniform resource locator (URL) and its structure. Whenever a user enters a URL in browser (Web Client), Web Server fetches that request and requested web page gets opened in browser. The Web Server uses a protocol called HTTP or Hyper Text Transport Protocol to communicate with Web Client. The standard used for creating hypertext documents for the WWW is Hyper Text Markup Language or HTML. We have also discussed the importance of web container and how it helps the programmer to deploy web application in container. We also discussed the tomcat server as a web container and its directory hierarchy to deploy the web application. So, in this unit we learnt important component of web application and container required to load java web application.

1.8 Answer for Check Your Progress

Check your progress 1: 1. b 2. c 3. c 4. c
Check your progress 2: 1. b 2. a
Check your progress 3: 1. d 2. a
Check your progress 4: 1. b 2. b 3. d
Check your progress 5: 1. a 2. a

1.9 Glossary

1. HTTP: It is the data communication protocol used to establish communication between client and server.
2. Container: It is used in java for dynamically generating the web pages on the server side.
3. Content-Type: It is HTTP header that provides the description about what are you sending to the browser.
4. URL: URL is the address of a document found on the WWW.
5. MIME: Multipurpose Internet Mail Extension (MIME) is a standard developed to expand the limited capabilities of email by providing support for varying content types and multi-part messages.

1.10 Assignment

1. Write short note on Client Server Environment.
2. Explain the role of HTTP in Client Server communication.
3. Explain the webapps directory of tomcat in detail.

1.11 Activities

Collect and learn some information on Http Protocol.

1.12 Case Study

- Analyse various Web container functionality and specification.

1.13 Further Readings

- https://docs.oracle.com/javaee/5/tutorial/doc/bnafe.html
- https://www.javatpoint.com/what-is-tomcat
- https://tomcat.apache.org/download-10.cgi
- https://codegym.cc/groups/posts/302-part-6-servlet-containers
- https://ecomputernotes.com/servlet/intro/servlet-container

UNIT 2: SERVLETS

Unit Structure

2.0 Learning Objectives

After learning this Unit, you will be:

- Able to define servlet and its life cycle
- Able to discuss different types of servlet
- Able to write servlet application to service client request
- Able to use different HTTP methods

2.1 Introduction

Web application means a software system that provides an interface to the user through a web browser. Examples of web applications include blogs, online e-commerce application, search engines etc. Web applications can be simple consisting of either static web pages or dynamic web pages and also interactive.

Initially, web server were using the CGI (Common Gateway Interface) to pass the request to an external program and after executing the program the content was sent back to the client as an output. Here in CGI, whenever the server receives a new request, it creates a new process to run the CGI program; which lead to creation of a new process resulting in to requirement of more server resources and time. Ultimately, it limits the number of requests that can be served concurrently. CGI applications are platform dependent.

To overcome all these issues Servlet came in. Static web pages are stored in the file system of web server usually displays the same information to all visitors. Whereas dynamic web pages are constructed through a business logic (a program) that produce the HTML content. This type of web application will provide personal information to the user and let them personalize the information according to their preferences. In this unit, we will discuss the Servlet as a web application development technique or language.

2.2 Servlet Basics

A Servlet is a java programming language used to extend the capabilities of servers that host applications and accessed as a request - response programming structure. Although, Servlet is a web component that is deployed on the web server to generate a dynamic web page based on the request.

Java Servlet is a part of Java Enterprise Edition (Java EE). The javax.servlet and javax.servlet.http package provides the various interfaces and classes for writing Servlet program.

There are mainly two types of servlets:

1. **Generic Servlet**: Generic servlets extend javax.servlet.GenericServlet class. Generic servlet is protocol independent servlet. It implements the Servlet and ServletConfig

interface. It may be directly extended by the servlet. Writing a servlet using GenericServlet class is very easy. It has only init() and destroy() method of ServletConfig interface in its life cycle. It implements the log method of ServletContext interface.

2. **Http Servlet**: HTTP servlets extend javax.servlet.HttpServlet class. HttpServlet is HTTP (Hyper Text Transfer Protocol) specific servlet. It provides an abstract class HttpServlet to the programmers to extend and create their own HTTP specific servlets.

Servlet Life Cycle

Every servlet has to pass through various life cycle stages. A servlet is basically a small Java program that executes within a Web server. It receives requests from clients and returns the responses back to the clients. The servlet life cycle is managed by the servlet container. The whole life cycle of a servlet breaks up into 3 phases as shown in figure 1.

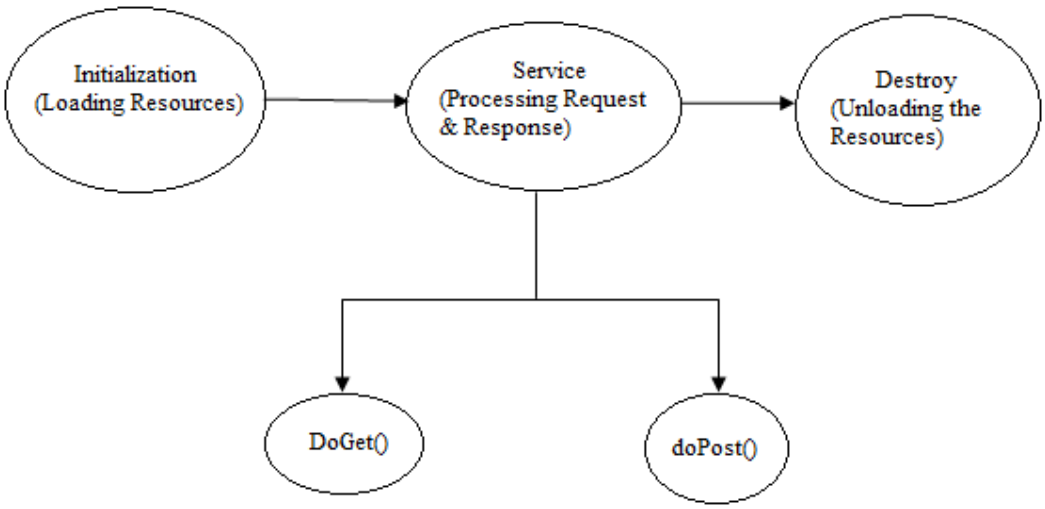


Figure 1. Servlet Lifecycle

Initialization stage: In this initial stage, a servlet is first loaded and initialized when it is requested by the corresponding clients.

Service stage: After initialization, the servlet serves the clients request, implementing the business logic of the web application. To serve the request, service methods invokes any one of GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS, TRACE and CONNECT.

Destruction stage: After all the pending requests are processed and the servlets have been idle for a specific amount of time, it is required to destroy them by the server and release all the resources they are occupying.

The behavior of a servlet is depicted in javax.servlet.Servlet interface, where the following methods are defined:

public void init(ServletConfig config) throws ServletException

This method is called once when the servlet is loaded into the servlet engine, before the servlet process its first request. The init method has a ServletConfig parameter as an argument. The servlet can read its initialization arguments through the ServletConfig object.

public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException

This method is invoked to process a request and service method invokes doGet, doPost, doPut, doDelete etc. methods as and when required. So user have nothing to do with service() method but user has to override either doGet() or doPost() method depending on the type of request received from the client. This method can be called zero one or many times until the servlet is unloaded. Once a servlet is loaded, it remains in the server's memory as a single object instance.

public void destroy():

This method is called only once thorough out the life cycle of servlet just before the servlet is unloaded and taken out of service.

Check your progress 1

1. What is the lifecycle of a servlet?

a. Servlet class is loaded

b. Servlet instance is created

c. init, service, destroy method is invoked

d. All of these

2. What are the functions of Servlet container?

a. Lifecycle management

b. Communication support

c. Multithreading support

d. All of the above

3. The life cycle of a servlet is managed by

a. Servlet context b. Servlet container c. The supporting protocol d. All of these

2.3 Servlet API

The Servlet API contains two important packages that provide all important classes and interface.

These are as follows:
1. javax.servlet
2. javax.servlet.http

Classes and Interfaces of javax.servlet package:

Interfaces	Classes
Servlet	GenericServlet
ServletConfig	HttpConstraintElement
ServletContext	HttpMethodConstraintElement
ServletContextListner	MultipartConfigElement
ServletRegistration	ServletContextEvent
ServletRequest	ServletInputStream
ServletRequestListner	ServletOutputStream
ServletResponse	ServletRequestAttributeEvent
ServletCookieConfig	ServletRequestEvent
SingleThreadModel	ServletRequestEvent
Filter	ServletRequestWrapper
FilterConfig	ServletResponseWrapper
FilterChain	ServletSecurityElement

Classes and Interfaces of javax.servlet.http package:

Interfaces	Classes
HttpServletRequest	Cookie
HttpServletResponse	HttpServlet

HttpSession	HttpServletRequestWrapper
HttpSessionActivationListner	HttpServletResponseWrapper
HttpSessionContext	HttpSessionBindingEvent
HttpSessionAttributeListner	HttpSessionEvent
HttpSessionIdListner	HttpUtils

Servlet Interface

Servlet interface defines some methods that all the servlet classes must implement. This method provides the following five methods. Out of these five methods, three methods are Servlet life cycle methods.

Servlet Interface Methods

Following are the methods of Servlet Interface:

Methods	Description
public void init(ServletConfig, config)	It is one of the Servlet life cycle methods. It is invoked by Servlet container after being initialized by Servlet.
public void service (ServletRequest req, ServletResponse res)	The **service( )** method is called after successful completion of **init( ).** It is invoked by Servlet container to respond to the requests coming from the client.
public ServletConfig getServletConfig( )	Returns a ServletConfig object, which contains initialization and startup parameters for this Servlet.
public String getServletInfo( )	Returns the information about the Servlet, such as author, version, and copyright. This method returns a string value.
public void destroy( )	Called by servlet container and it marks the end of the life cycle of a servlet. It indicates that servlet has been destroyed.

HttpServlet Class

The HttpServlet class extends the GenericServlet and implements Serializable interface. It is an abstract class. The HttpServlet class reads the HTTP request from http, get, post, put, delete etc. It calls one of the corresponding methods.

HttpServlet Class Methods

- protected void doGet(HttpServletRequest req, HttpServletResponse resp)
- protected void doDelete(HttpServletRequest req, HttpServletResponse resp)
- protected void doHead(HttpServletRequest req, HttpServletResponse resp)
- protected void doPost(HttpServletRequest req, HttpServletResponse resp)
- protected void doPut(HttpServletRequest req, HttpServletResponse resp)
- protected void doTrace(HttpServletRequest req, HttpServletResponse resp)
- protected void service(HttpServletRequest req, HttpServletResponse resp)

GenericServlet Class

It is an abstract class that implements Servlet, ServletConfig and serializable interface. It provides the implementation of all methods of these interfaces except the service method. GenericServlet may be directly extended by Servlet. It provides simple versions of the life cycle methods init( ) and destroy( ) methods.

GenericServlet Class Methods

Following are the important methods of GenericServlet Class:

Methods	Description
public void destroy( )	Invoked by servlet container. It shows that the servlet is being taken out of service.
public String getInitParameter(String name)	Returns a String containing the value of named parameter.
public String getServletInfo( )	Returns information related to Servlet like author, version etc.
public String getServletName( )	Returns the name of Servlet object.
public void init( )	It is a convenience method that can easily be overridden so that we do not need to call **super.init(config).**
public void log(String msg)	Writes the given message to a Servlet log file.

public abstract void service(ServletRequest req, ServletResponse res)	It is an abstract method, called by the servlet container to allow the servlet to respond to a request.

Check your progress 2

1. Which packages represent interfaces and classes for servlet API?

a. javax.servlet

b. javax.servlet.http

c. Both a & b

d. None of these

2. Which type of ServletEngine is a server that includes built-in support for servlets?

a. Standalone ServletEngine

b. Embedded ServletEngine

c. Add-on ServletEngine

d. None of these

2.4 Servlet Creation in Tomcat

To write a Servlet application we need to follow the certain steps. These steps are common for all kinds of Web server. Apache Tomcat is an open source web server for testing servlets and JSP technology. So, we will use Apache Tomcat to test all the Servlet and JSP applications. Follow below mentioned steps to develop Servlet.

- Create directory structure for the application
- Create a Servlet
- Compile the Servlet
- Create the deployment descriptor of the application
- Start the server and deploy the application

Creating the Directory Structure

There is a unique directory structure that must be followed to create Servlet application. This structure tells the developer regarding where to put different types of files.

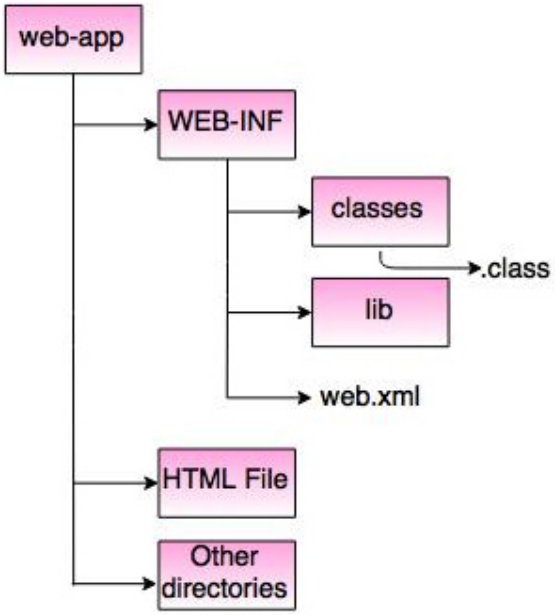


Figure 2. Directory Structure of Servlet Application

Create a Servlet

```
//ServletTest.java
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class ServletTest extends HttpServlet
{
   public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
   {
      res.setContentType("text/html");
      PrintWriter pw = res.getWriter();
      pw.println("<html><body>");
      pw.println("<h2>");
      pw.println("You are welcome to BAOU, Ahmedabad");
      pw.println("</h2>");
      pw.println("</body></html>");
      pw.close();
```

```
    }
}
```

PrintWriter is an abstract class for writing to character streams. Methods to be implemented are write(char[], int, int), flush() and close(). getWritet() method of HttpServletResponse returns the PrintWriter. PrintWriter object will be used to send character text to the client.

Compile the Servlet program

Assuming the classpath and environment is setup properly.

Create a deployment descriptor

The deployment descriptor is an xml file. It is used to map URL to servlet class, defining error page.

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_9" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee'
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd">
<display-name>Servlet Example</display-name>
    <servlet>
      <servlet-name>BAOU</servlet-name>
      <servlet-class>ServletTest</servlet-class>
   </servlet>
   <servlet-mapping>
       <servlet-name>BAOU</servlet-name>
       <url-pattern>/welcome</url-pattern>
   </servlet-mapping>
</web-app>
```

Save the web.xml file in <Tomcat-installation directoryt>/webapps/ROOT/WEB-INF/

- Now start the Tomcat server

- Open browser and type

  http://localhost:8080/

  Click on /BAOU webApplication.
- It will execute our servlet and display following output in browser.

Figure 3. Tomcat output

Check your progress 3

1. _____method obtains a byte-based output stream that enables binary data to be sent to the client.

a. sendRedirect()

b. getOutputStream()

c. getOutput()

d. getWirter()

2. _____ method obtains a character-based output stream that enables text data to be sent to the client.

a. getWirter()

b. getOutput()

c. getOutputStream()

d. sendRedirect()

2.5 HTTP Methods

HttpServlet class extends the GenericServlet class which is protocol-dependent. HttpServlet class is in javax.servlet.http package. A subclass of HttpServlet must override a minimum one method from the following methods:

1. public void service(ServletRequest req, ServletResponse res): This method will dispatch the requests to the protected service method. Before dispatching request it converts the request and response object into http type.

Syntax:

public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException

2. protected void service(HttpServletRequest req, HttpServletResponse res): This method receives HTTP requests from the public service method and dispatches the request to the doXXX methods defined in HTTP class.

Syntax:

protected void service(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException

3. protected void doGet(HttpServletRequest req, HttpServletResponse res): This method is called by web container for handling GET requests.

Syntax:

protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,IOException

4. protected void doPost(HttpServletRequest req, HttpServletResponse res): This method is called by web container for handling POST requests.

Syntax:

protected void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException,IOException

5. protected void doHead(HttpServletRequest req, HttpServletResponse res): This method is called by web container for handling HEAD requests.

Syntax:

protected void doHead(HttpServletRequest req, HttpServletResponse res) throws ServletException,IOException

6. protected void doOptions(HttpServletRequest req, HttpServletResponse res): This method is called by web container for handling OPTIONS requests.

Syntax:

protected void doOptions(HttpServletRequest req, HttpServletResponse res) throws ServletException,IOException

7. protected void doPut(HttpServletRequest req, HttpServletResponse res): This method is called by web container for handling PUT requests.

Syntax:

protected void doPut(HttpServletRequest req, HttpServletResponse res) throws ServletException,IOException

8. protected void doTrace(HttpServletRequest req, HttpServletResponse res): This method is called by web container for handling TRACE requests.

Syntax:

protected void doTrace(HttpServletRequest req, HttpServletResponse res) throws ServletException,IOException

9. protected void doDelete(HttpServletRequest req, HttpServletResponse res): This method is called by web container for handling DELETE requests.

Syntax:

protected void doDelete(HttpServletRequest req, HttpServletResponse res) throws ServletException,IOException

10. protected long getLastModified: This method returns the time the HttpServletRequest object was last modified in milliseconds. It will return a negative number if time is unknown.

Syntax:

protected long getLastModified(HttpServletRequest req)

Check your progress 4

1. Which class can handle any type of request so that it is protocol-independent?

a. GenericServlet

b. HttpServle

c. Both a & b

d. None of these

2. What type of servlets use these methods doGet(), doPost(),doHead, doDelete() and doTrace()?

a. HttpServlets

b. Genereic Servlets

c. All of these

d. None of these

2.6 Differences between GET and POST

The HTTP POST requests supply additional data from the client (browser) to the server in the request body. While, GET requests include all required data in the URL. Forms in HTML can use either of the method by specifying method="POST" or method="GET" in the <form> element ("GET" is default). The method specified will determine how form data will be submitted to the server. So, let's understand the behaviour of both the methods through some important features.

Security:

GET method is less secure compared to POST because data sent is part of the URL, So it's saved in browser history. POST method is a little safer than GET because the parameters are not stored in browser history.

Re-submit behaviour:

GET requests are re-executed but may not be re-submitted to server if the HTML is stored in the browser cache memory. While in POST, the browser usually alerts the user that data will need to be re-submitted.

Form data type:

In GET, only ASCII characters allowed, while in POST No restrictions is there as binary data is also allowed.

Usability:

GET method should not be used when sending password like or other sensitive information. While, POST method can be used to send passwords or other sensitive information.

Visibility:

GET method is visible to everyone as it will be displayed in the browser's address bar and has limits on the amount of information to send. While, POST method data are not displayed in the URL.

Caching:

Get requests can be cached while POST requests can not be cached.

Check your progress 5

1. What type of servlets use doGet(), doPost(),doHead, doDelete(), doTrace() methods?

a. Genereic Servlets

b. HttpServlets

c. All of these

d. None of these

2. Which HTTP Request method is non-idempotent?

a. GET

b. POST

c. Both a & b

d. None of the above

3. Which object is created by the web container at time of deploying the project?

a. ServletContext

b. ServletConfig

c. HttpServlet

d. None of these

2.7 Let Us Sum Up

In this unit, we learnt that how a Servlet gets created and from which stages it passes throughout its journey to serve the clients request. Apart from its life cycle, we also discussed the steps to create a servlet and its deployment in the web server i.e Tomcat Server.

We also discussed various classes and interfaces of Servlet through its API. An HTTP Servlet handles client requests through its service method which indirectly invokes required getXXX() methods to serve the requests. At last, we discussed the important difference between the GET and POST methods of HttpServlet.

2.8 Answer for Check Your Progress

Check your progress 1: 1. d 2. d 3. b
Check your progress 2: 1. c 2. a
Check your progress 3: 1. b 2. a
Check your progress 4: 1. a 2. a
Check your progress 5: 1. b 2. c 3. a

2.9 Glossary

1. HTTP: It is the data communication protocol used to establish communication between client and server.

2. Deployment Descriptor: A web application's deployment descriptor is responsible to map the http request with the specific servlets. Whenever the web server receives a request for the

application, it uses the deployment descriptor to map the URL of the request to the servlet available to handle the request. The name of the deployment descriptor should be web.xml.

3. HTTP Requests: It is the request sent by the web client (browser) to a web server that contains all sorts of resources.

4. CGI: The Common Gateway Interface (CGI) is one of the techniques used for generating dynamic content in web applications.

2.10 Assignment

1. Write short note on different Http Methods.
2. Explain Servlet life cycle.
3. Differentiate between Get and Post method.
4. Discuss the importance of service() method of HttpServlet.

2.11 Activities

Write a servlet application to use serveltConfig and servletContext.

2.12 Case Study

- Anatomy of various HTTP methods.
- Analyse the HttpRequest and HttpResponse headers.

2.13 Further Readings

- https://docs.oracle.com/javaee/5/tutorial/doc/bnafe.html
- https://www.javatpoint.com/get-vs-post
- https://www.studytonight.com/servlet/
- https://www.scaler.com/topics/difference-between-get-and-post/
- https://www.w3schools.com/tags/ref_httpmethods.asp

UNIT 3: SERVLET COLLABORATION AND CONFIGURATION

Unit Structure

3.0 Learning Objectives

After learning this Unit, you will be:

- Able to differentiate servletcontext and servletconfig
- Able to use servletcontext and servletconfig in applications
- Able to redirect request to other servlet
- Able to pass attributes to servlet in application

3.1 Introduction

As we have seen in previous unit that, a servlet allows us to provide a dynamic programming in web application using Java. In this unit we are going to explore the importance of ServletConfig and ServletContext to configure the servlet in an application. If there are more than one servlet in a web application and they wants to communicate with each other then we need a mechanism which allow us to do so. Request Dispatcher class will allow us this functionality to make communication possible between different servlets in web application. We have also discussed regarding setting and accessing attribute in servlet.

3.2 Servlet Config

ServletConfig is an interface contained in javax.servlet package as a part of servlet API. For every Servlet class in web application, one ServletConfig object will be created by the web container and this object will be passed as an argument to the public void init(ServletConfig config) method of our Servlet class object by the web container. ServletConfig is an object contains configuration information created by Servlet Container and will be passed to the servlet during initialization. ServletConfig is meant for a specific servlet, means one can store servlet-specific information in web.xml file. We can retrieve the initialization parameters from web.xml through ServletConfig.getInitParam("paramName") method.

The below table shows various methods of the ServletConfig interface:

Methods	**Details**
java.lang.String getInitParameter(String name)	This method returns a String containing value of specified name initialization parameter, or null if the parameter does not exist.
java.util.Enumeration getInitParameterNames()	This method returns a java.util.Enumeration of String objects containing

	the servlet's initialization parameters names, or an empty java.util.Enumeration if servlet has none.
ServletContext getServletContext()	This method returns a reference to the ServletContext in which the caller is executing.
java.lang.String getServletName()	This method returns name of this servlet instance.

Example: In the following example we have passed the name and address in the web.xml file. In the ServletTest file we are accessing those parameters using servletconfig object.

ServletTest.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class ServletTest extends HttpServlet
{
   private ServletConfig config;
 public void init(ServletConfig config)
 {
      this.config=config;
  }
 public void service(HttpServletRequest req, HttpServletResponse res)throws
ServletException, IOException
          {
      String username=config.getInitParameter("name");
      String sarnamu=config.getInitParameter("address");
      res.setContentType("text/html");
      PrintWriter pw = res.getWriter();
                  pw.println("<html><body>");
                  pw.println("<h2>");
                  pw.println("You are welcome to BAOU, Ahmedabad");
                  pw.println("</h2>");
                  pw.println("<h2>" + "Servlet Example" + "</h2>");
                  pw.println("Welcome " + "<b>" + username + "</b> </br>");
                  pw.println("<b>" + sarnamu + "</b> </br>");
```

```
                pw.println("</body></html>");
                pw.close();
    }
}
```

Web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
 id="WebApp_ID" version="4.0">
 <display-name>Servlet Example</display-name>
    <servlet>
      <servlet-name>BAOU</servlet-name>
      <servlet-class>ServletTest</servlet-class>
                <init-param>
        <param-name>name</param-name>
        <param-value>ved</param-value>
      </init-param>
                <init-param>
        <param-name>address</param-name>
        <param-value>Gandhinagar</param-value>
      </init-param>
  </servlet>
  <servlet-mapping>
       <servlet-name>BAOU</servlet-name>
       <url-pattern>/welcome</url-pattern>
   </servlet-mapping>
</web-app>
```

Output:

Figure 1: Use of ServletConfig to access initialization parameter

Check your progress 1

1. Deployment Descriptor (DD) is used for initializing parameter.

a. True

b. False

2. When the Web Container initializes a servlet, it creates a _____ object for the servlet?

a. ServletConfig

b. ServletInit

c. ServletContext

d. None of these

3.3 Servlet Context

ServletContext interface is used to provide the configuration information per web application within the servlet container (like Tomcat, Glassfish). This interface can be used to provide information for more than one servlet of web application from web.xml file. This context is defined in javax.servlet.ServletContext interface in a Servlet API. Different servlets deployed in the same webapp can share this information between them using the shared ServletContext object.

For sharing this information from web.xml user need to pass it in <context-param> tag having sub tags, <param-name> contains the name and <param-value> tag contains value. This information can be retrieved through ServletConfig.getServletContext() method.

There are three ways to obtain an object of ServletContext interface:

Way 1:

- ServletConfig conf = getServletConfig();
- ServletContext context = conf.getServletContext();

In this approach, first obtain an object of ServletConfig interface and using it call getServletContext() to get Context object.

Way 2:

In this approach, you need to just call getServletContext() method of GenericServlet class.

```
public class BAOU extends HttpServlet
{
        public void doGet / doPost(HttpServletRequest req, HttpServletResponse res)
        {
                ServletContext context = getServletContext();
        }
}
```

Way 3:

In this approach, you can get the object of ServletContext by making use of HttpServletRequest object.

```
public class BAOU extends HttpServlet
{
        public void doGet/doPost(HttpServletRequest req, HttpServletResponse res)
        {
                ServletContext context = req.getServletContext();
        }
}
```

The table below shows the declarations of some of the more common methods in the ServletContext interface we use on the site:

Methods	Details
java.lang.Object getAttribute(String name)	This method returns the attribute of specific name.
java.util.Enumeration getAttributeNames()	This method returns an java.util.Enumeration containing all attribute names available within this servlet context.
java.util.Enumeration getInitParameterNames()	This method returns a java.util.Enumeration of String objects containing the servlet

	context initialization parameters names, or an empty java.util.Enumeration if servlet context has none.
RequestDispatcher getRequestDispatcher(String path)	This method returns a RequestDispatcher object that works as a wrapper for the resource located at specified path.
void removeAttribute(String name)	With this method we can remove the attributes of name using this method.
void setAttribute(String name, Object object)	This method binds an object to a specified name attribute in this servlet context.

Example: In the following example we have passed the Databse driver and databse name in the context-param tag of web.xml file. In the ServletTest file we are accessing those parameters using servletContext object.

ServletTest.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import javax.servlet.ServletContext;
import java.io.*;
public class ServletTest extends HttpServlet
{
   private ServletConfig config;
   private ServletContext context;
   public void init(ServletConfig config)
{
      this.config=config;
}
public void doGet(HttpServletRequest req, HttpServletResponse res)throws ServletException, IOException
 {
      context=req.getServletContext();
      String dvr=context.getInitParameter("Driver");
      String dbname=context.getInitParameter("Database");
      res.setContentType("text/html");
      PrintWriter pw = res.getWriter();
                  pw.println("<html><body>");
```

```
                pw.println("<h2>");
                pw.println("You are welcome to BAOU, Ahmedabad");
                pw.println("</h2>");
                pw.println("<h2>" + "ServletContext Example" + "</h2>");
                pw.println("Driver user: " + "<b>" + dvr + "</b> </br>");
                pw.println("Database Name:" + "<b>" + dbname + "</b> </br>");
                pw.println("</body></html>");
                pw.close();
  }
}
```

Web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
 id="WebApp_ID" version="4.0">
 <display-name>ServletContext Example</display-name>
   <context-param>
        <param-name>Driver</param-name>
        <param-value>com.mysql.Driver</param-value>
   </context-param>
         <context-param>
        <param-name>Database</param-name>
        <param-value>MCA</param-value>
   </context-param>
         <servlet>
      <servlet-name>BAOU</servlet-name>
      <servlet-class>ServletTest</servlet-class>
          </servlet>
  <servlet-mapping>
       <servlet-name>BAOU</servlet-name>
       <url-pattern>/welcome</url-pattern>
```

```
  </servlet-mapping>
</web-app>
```

Output:

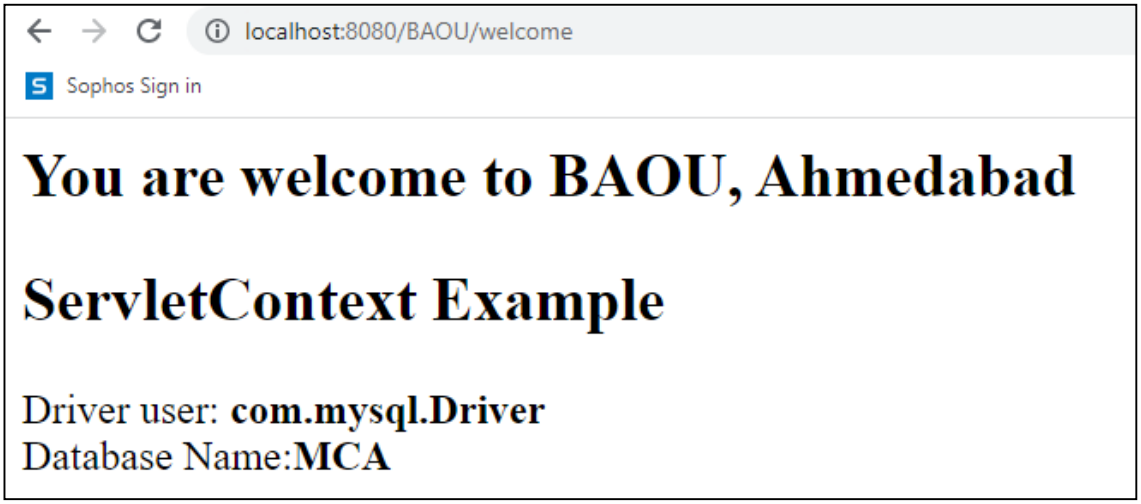


Figure 2: Use of ServletContext to access initialization parameter

Check your progress 2

1. _____ object is available to any servlet or JSPs that are part of the web app and provides communication between servlets and JSPs.

a. Servlet

b. ServletConfig

c. ServletContext

d. HttpServletContext

2. Which object is created by the web container at time of deploying the project?

a. ServletConfig

b. ServletContext

c. Both a & b

d. None of these

3.4 Request Dispatcher

RequestDispacther is an interface used to receive requests from the users and dispatches it to other resource files like HTML file, Servlet file, JSP file etc. Servlet container is responsible

to generate RequestDispatcher object. RequestDispacther provides forward() and include() methods. These methods are used to call RequestDispacther.

This interface has following two methods:

- public void forward(ServletRequest request, ServletResponse response):
  When this method is called then the request of current file is send forward to the another resource such as servlet, JSP, HTML file etc. and the response of that file will be provided by the server.
- public void include(ServletRequest request, ServletResponse response):
  When this method is called then the content of current file such as HTML, Servlet, JSP is included with the response.

Difference between forward() vs include() :

Include()	Forward()
This method Includes another file in our current file.	This method will forward the client request to the forwarding page.
Control from client is temporary shifted .	Control from client is permanently shifted to forwarded file.
Once the control is returned to the client, any activity such as calling another servlet or another RequestDispatcher object can be performed.	Once the control is returned to the client the output can not be modified.
It is generally used to include other web resource such as banner contents, copyright information and so on.	It is generally used when further processing is given to another web resource.
Speed of delivery to client is slower to execute.	Speed of delivery to client is faster to execute.
Here, the client receives the response from the same servlet which he has requested.	Here, the client actually receives the response from a different servlet (which is not known to client)
It is used when static information is to be included.	It is used when dynamic information is to be included, where a Servlet has to play the role of a controller to process the client input and deciding the response to be returned.

Getting RequestDispatcher

RequestDispatcher can be obtained from a request object or from a servlet context.

- RequestDispatcher dispatcher = request.getRequestDispatcher("ved.jsp");
  dispatcher.forward(request, response);

We can get the RequestDispatcher from the request object with the getRequestDispatcher() method.

- RequestDispatcher dispatcher =
  getServletContext().getRequestDispatcher("/ved.jsp");
  dispatcher.forward(request, response);

Here we get the RequestDispatcher from the servlet context. In this case, the path must begin with a slash character.

Example:

In this example, we are using both the methods include and forward. Using include method, we will be changing the content of current page and when we are ready to transfer the control to the next page, we will use forward method. Here we are using index.html to get username and password from the user, Validation Servlet will validate the password entered by the user, if the user has entered username as "ved" and password "desai" then he will be forwarded to WelcomeTest Servlet else the user will stay on the index.html page and an error message will be displayed.

index.html:

```
<html>
<form action="loginPage" method="post">
 User Name:<input type="text" name="uname"/><br/>
 Password:<input type="password" name="upass"/><br/>
 <input type="submit" value="SUBMIT"/>
</form>
</html>
```

Validation.java:

```
import java.io.*;
import java.io.*;
import javax.servlet.*;
```

```
import javax.servlet.http.*;
public class Validation extends HttpServlet
{
  public void doPost(HttpServletRequest request,
    HttpServletResponse response)
      throws ServletException, IOException
  {
      response.setContentType("text/html");
      PrintWriter pw = response.getWriter();
      String name=request.getParameter("uname");
      String pass=request.getParameter("upass");
      if(name.equals("ved") &&  pass.equals("desai"))
      {
        RequestDispatcher dis=request.getRequestDispatcher(“welcome”);
        dis.forward(request, response);
      }
      else
      {
        pw.print(“User name or password is incorrect!”);
        RequestDispatcher dis=request.getRequestDispatcher(“index.html”);
        dis.include(request, response);
      }
  }
}
```

WelcomeTest.java:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class WelcomeTest extends HttpServlet {

 public void doPost(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException
 {
```

```
   response.setContentType("text/html");
   PrintWriter pw = response.getWriter();

   String name=request.getParameter("uname");
   pw.print("Hello"+name+"!");
   pw.print(" Welcome to BAOU");
}
}
```

web.xml:

```
<web-app>
<display-name>BAOU-Ahmedabad</display-name>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
</welcome-file-list>
 <servlet>
<servlet-name>Login</servlet-name>
<servlet-class>Validation</servlet-class>
</servlet>
<servlet>
<servlet-name>Welcome</servlet-name>
<servlet-class>WelcomeTest</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Login</servlet-name>
<url-pattern>/loginPage</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>Welcome</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
</welcome-file-list>
</web-app>
```

Output:

Entering wrong credentials:

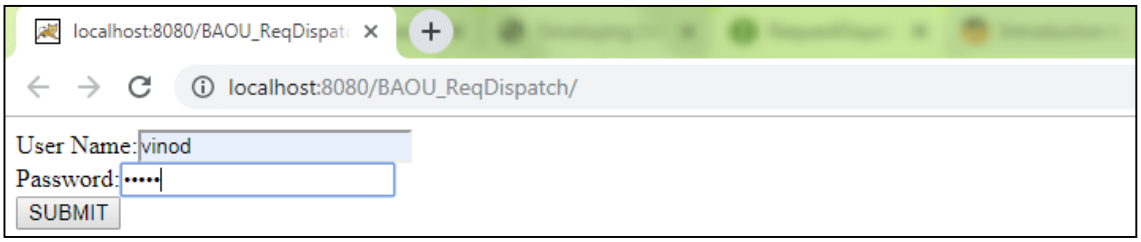


Figure 3: Asking for credentials

Error screen:

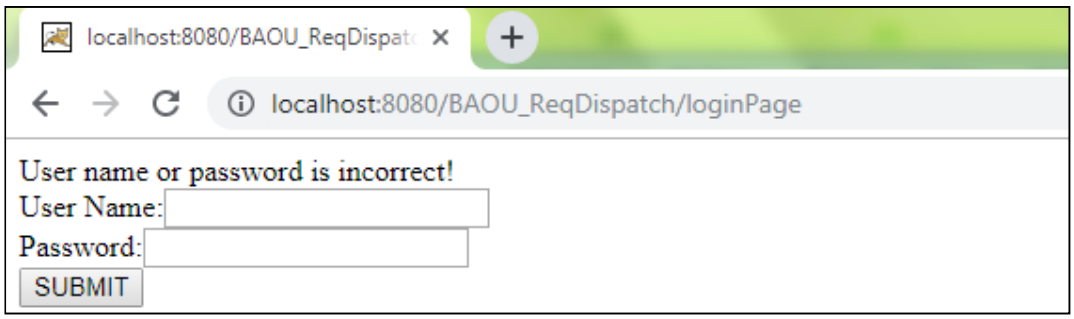


Figure 4: Redirected to login page on wrong credentials

Welcome screen on entering correct user name and password:

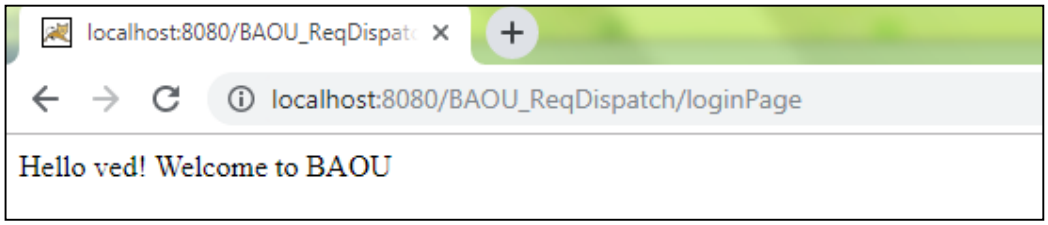


Figure 5: Redirection to welcome page

Check your progress 3

1. Which method defined in the HttpServletRequest returns the object of RequestDispatcher ?

a. getRequestDispatcher()

b. getDispatcher()

c. getRequest()

d. requestDispatcher()

2. Which statement is true about include() method of RequestDispatcher interface ?

a. forwards a request from a servlet to another resource on the server

b. includes the content of any resource inside the current servlet

c. includes the content of only servlet inside the current servlet

d. None

3.5 Send Redirect

This method is exposed by HttpServletResponse interface. This method redirects the request to completely other resource existing on different server or context. Its syntax is:

- public void sendRedirect(String path) throws java.io.IOException

It requires path as the url of the destination resource. The method is called using the HttpServletResponse as shown below:

- response.sendRedirect("https://www.spuvvn.edu/");

When this method is called, the server sends back a HTTP status code of 302 (temporary redirection) which forces the web browser creates a new HTTP GET request for the content at the redirected path. It is generally used when user want to use an external resource (available outside the server) to complete the processing of the request. This method should be called before sending the response or otherwise it will throw an IllegalStateException.

The real time use case of this method is, when a customer makes a payment for items in e-commerce website, the customer is always redirected to external merchant site for completing the payment.

Following steps are the process flow of this method:

- First the client sends a HTTP request to some.jsp
- Then the server sends a HTTP response back with path: other.jsp in the header.
- Again the client sends a HTTP request to other.jsp (will be visible in the browser address bar)
- At last, the server sends a HTTP response back with content of other.jsp

Example: In this example we are redirecting the user to the BAOU website.

```
import javax.servlet.http.*;
import javax.servlet.*;
import javax.servlet.ServletContext;
import java.io.*;
public class ServletTest extends HttpServlet
{
```

```
   public void doGet(HttpServletRequest req, HttpServletResponse res)throws
ServletException, IOException
          {
       res.setContentType("text/html;charset=UTF-8");
      PrintWriter pw = res.getWriter();
                  res.sendRedirect("https://baou.edu.in/");
      pw.println("<html><body>");
                  pw.println("You are redirected to BAOU, Ahmedabad");
                  pw.println("</body></html>");
                  pw.close();
   }
}
```

Check your progress 4

Which method of HttpServletResponse is used to redirect an HTTP request to another URL?

a. sendURL()

b. redirectURL()

c. sendRedirect()

d. getRequestDispatcher()

3.6 Working with Attributes

An attribute is an object. Using this attribute user can share information in a web application. Attribute provides functionality to Servlets to share information among themselves. The servlet programmer can transmit information from one servlet to another servlet using attributes. It is same as passing an object from one class to another class so that we can reuse the same object.

We can SET and GET attributes from one of the following scopes:

- request
- session
- application

The syntax to SET and GET an Attribute in servlet is as follows:

- public void setAttribute(String str, Object obj) method is used to SET an Attribute.
- public Object getAttribute(String str) method is used to GET an Attribute.

Example of Setting and getting an Attribute:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletTest extends HttpServlet
{
 protected void doGet(HttpServletRequest req, HttpServletResponse res)
          throws ServletException, IOException
{
       res.setContentType("text/html");
       PrintWriter out = res.getWriter();
       ServletContext context = getServletContext();
       context.setAttribute("user","vinod");
       String str = (String)context.getAttribute("user");
        out.println("Welcome Mr.:" + str);
        out.close();
   }
}
```

Output:

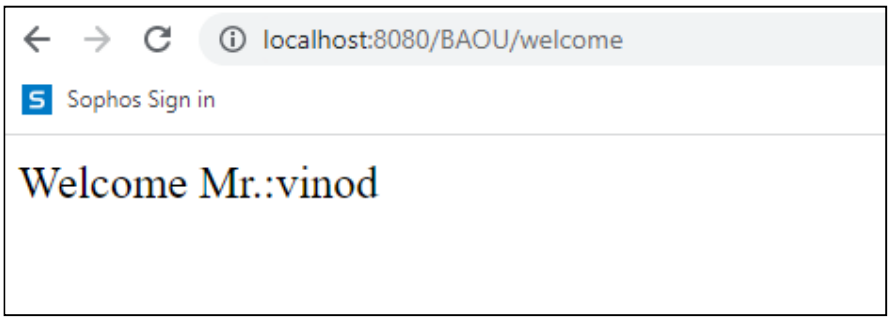


Figure 6: Set and Get Attribute in Servlet

Check your progress 5

From the following which types of objects can store attributes?

a. ServletConfig

b. ServletResponse

c. RequestDispatcher

d. HttpServletRequest

3.7 Let Us Sum Up

In this unit we have learnt that through ServletConfig and ServletContext we can configure parameter for individual Servlet or for whole web application. RequestDispatcher is an interface, which defines an object that can dispatch request to any resources on the server. Basically, a forward request can be used if the operation can be safely repeated upon a browser reload of the resulting web page; otherwise, redirect is preferable. Typically, if the task require to perform an edit operation on the data store, then a redirect is preferable not a forward method. If resources to be included in the same page then include method is preferable. At last, we have discussed the setting and accessing attributes in the Servlet from a particular scope.

3.8 Answer for Check Your Progress

Check your progress 1: 1. a 2. a

Check your progress 2: 1. c 2. b

Check your progress 3: 1. a 2. b

Check your progress 4: c

Check your progress 5: d

3.9 Glossary

1. ServletConfig: It is an object available one per Servlet component or class.

2. ServletContext: It is an object which is available one per web application.

3. Redirect: This terminology indicates that the response will be redirected to another resources such as jsp, servlet, html file.

3.10 Assignment

1. What is difference between ServletConfig and ServletContext?
2. How do we call one servlet from another servlet?

3. How can we invoke another servlet in a different application?
4. What is difference between ServletResponse sendRedirect() and RequestDispatcher forward() method?

3.11 Activities

- Explore different methods of ServletConfig and ServletContext and its working.

3.12 Case Study

- Anatomy of an RequestDispatcher forward() and include() methods.

3.13 Further Readings

- https://docs.oracle.com/javaee/5/tutorial/doc/bnafe.html
- https://www.geeksforgeeks.org/servletconfig-in-servlet/
- https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaServlets.html

UNIT 4: SESSION MANAGEMENT

Unit Structure

4.0 Learning Objectives

After learning this Unit, you will be:

- Able to define stateful and stateless object
- Able to define session
- Able to define cookie and HttpSession
- Able to write servlet to pass data between servlets

4.1 Introduction

HTTP protocol and Web Servers are stateless means that every request to a web server is a new request to process and they can't identify whether it's coming from the same client or not. But sometimes it becomes very important to know who the client is in the web applications and process the request. Therefore, it is important to keep track of a client's current status to recognize them. The time interval in which the client and the server communicate with each other is known as a session.

4.2 Session and its Importance

A session is a mechanism of keeping record of different activities across multiple requests made by a single client. Today, Session management is a crucial feature of all modern web applications which allows the server to track and remember its clients. By keeping a session for each client, the Server can serve the client's request in a much better way. It also helps in safety, security and personalization of web applications. Today, all modern e-commerce related web applications like Amazon, Flip Cart or e-bay manages session of every client to store their activities since they logs in till logged out. We can have following benefits from the application if session is maintained:

- It helps to maintain user status and data to all over the application.
- It can be easily implemented and allows for storing any kind of object like dataset.
- It allows for storing every client data separately.
- It is secure and transparent from user because session object is stored on the server.

Check your progress 1

Which of the following is wrong about session?

a. Default timeout value for session variable is 20 minutes

b. All users have same session variable

c. All users connect to the same session

d. New session cannot be created for a new user

4.3 Information Passing between Client and Server

There are several ways through which we can pass information between client and server through request and response. The Servlet application provides four unique session tracking approaches. These are HttpSession, Cookies, Hidden Form Field and URL rewriting.

1. **User Authentication**: This is the very common way where we user can provide authentication credentials from the login page and then we can pass the authentication information between server and client to maintain the session. This is not very effective method because it won't work if the same user is logged in from different browsers.
2. **Hidden Form Field:** We can create a unique hidden field in the HTML and when user starts navigating, we can set its value unique to the user and keep track of the session. This method can't be used with links because it needs the form to be submitted every time request is made from client to server with the hidden field. Also it's not secure because we can get the hidden field value from the HTML source and use it to hack the session.
3. **URL Rewriting**: We can append a session identifier parameter with every request and response to keep track of the session. This is very tedious because we need to keep track of this parameter in every response and make sure it is not clashing with other parameters. For example:

   Original URL: http://server:port/baou/ServletName

   Rewritten URL: http://server:port/baou/ServletName?sessionid=1234

   This session tracking mechanism does not require any special support from the browser. At the same time the drawback of this technique is, it is tedious to implement for session tracking.
4. **Cookies**: Cookies are small piece of information that is sent by web server in response header and gets stored in the browser cookies. When client make further request, it adds the cookie to the request header and we can utilize it to keep track of the session. A cookie has a name, a single value, expiration date and optional attributes. A cookie's value can uniquely identify a client. Since a client can disable cookies, this is not the most secure and fool-proof way to manage the session. We can

maintain a session with cookies but if the client disables the cookies, then it won't work. If Cookies are disabled then we can fallback to URL rewriting to encode Session id e.g. JSESSIOINID into the URL itself.

5. **Session Management API**: Session Management API is built on top of above methods for session tracking. HttpSession object is used to store entire session with a specific client. We can store, retrieve and remove attribute from HttpSession object. Any servlet can have access to HttpSession object throughout the getSession() method of the HttpServletRequestobject.

Some of the major disadvantages of all the above methods are:

- Most of the time we don't want to only track the session, we have to store some data into the session that we can use in future requests. This will require a lot of effort if we try to implement this.

All the above methods are not complete in themselves; all of them will not work in a particular scenario. So we need a solution that can utilize these methods of session tracking to provide session management in all cases.

Check your progress 2

1. Which of the below is not a session tracking method?

a. URL rewriting

b. History

c. Cookies

d. SSL sessions

2. Which of the following is stored at client side?

a. URL rewriting

b. Hidden form fields

c. SSL sessions

d. Cookies

4.4 Cookies

Cookies are small piece of information on the client computer to store the client state. It is a key value pair of information, sent by the server to the browser and then browser sends back this identifier to the server with every subsequent request. There are two types of cookies:

1. **Session or Non-persistent** cookies are temporary cookies and are deleted as soon as user closes the browser. They are valid for a single session only. The persistent cookies are live till the browser is open.
2. **Persistent cookies** have expiry time. They remains valid for multiple sessions. They are not deleted when user closes the browser. They are stored in primary memory of the system. They are invalidated when user logs out or signs out.

Whenever a cookie is associated with the client request, server will associate it with subsequent user session otherwise server will create a new unique cookie and send it back with response. Following is the simple code to create a cookie with name "baou" and a value:

- Cookie cukie = new Cookie("baou", "value");

The servlet sends cookies to the browser using following code:

- HttpServletResponse.addCookie(javax.servlet.http.Cookie)

The major disadvantage of this approach is that whenever a user disables cookie support in a browser in that case server will not be able to identify the user.

Constructors:

Constructor	Details
Cookie( )	It constructs the cookie with default property.
Cookie(String name, String value)	It constructs a cookie with specified name and value.

Methods of Cookie:

Following are the methods supported by Cookie class:

Methods	Details
public String getName( )	It returns the name of the cookies.
public String getPath( )	It returns the path of the server to which the browser returns the cookie.
public String getValue( )	It returns the value of the cookie.
public int getMaxAge( )	It returns the maximum age limit to the cookie in seconds.
public void setMaxAge(int expiry)	It sets the maximum age of the cookies in seconds.
public void setValue(String newValue)	It allocates a new value to a cookie after the cookie is created.

Deleting the Cookies

We can delete the cookies from the browser by setting the cookie expiry time to 0 or -1.

Example

- cookie.setMaxAge(0);

Example: In the following example we are storing and retrieving cookie data.

index.jsp

```
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  <title>Servlet Example</title>
 </head>
 <body>
  <h1>Welcome to BAOU</h1>
        <h1>Cookie Example</h1>
  <form action="login">
                User Name:<input type="text" name="user"/></br>
                Password:    <input type="password" name="Pass"/></br>
                <input type="submit" value="Login"/>
        </form>
 </body>
</html>
```

ServletTest.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletTest extends HttpServlet
{
        public void doGet(HttpServletRequest req, HttpServletResponse res)
          {
    try{
       res.setContentType("text/html");
       PrintWriter out = res.getWriter();
```

```
        String name = req.getParameter("user");
        String password = req.getParameter("Pass");
        out.print("Hello : "+name);
                    out.print("</br>");
        out.print("Your Password is : "+password);

        //Creating two cookies
        Cookie c1=new Cookie("userName",name);
        Cookie c2=new Cookie("Password",password);

        //Adding the cookies to response header
        res.addCookie(c1);
        res.addCookie(c2);
        out.print("</br>");
        out.print("<a href='welcome'>Show Details</a>");
        out.close();
  }
  catch(Exception e)
  {
     System.out.println(e);
   }
 }
}
```

ServletTest1.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletTest1 extends HttpServlet
{
 public void doGet(HttpServletRequest req, HttpServletResponse res)
 {
   try{
      res.setContentType("text/html");
```

```
    PrintWriter out = res.getWriter();

    //Reading cookies
    Cookie c[]=req.getCookies();
    //Displaying User name value from cookie
    out.print("Name : "+c[1].getValue());
    //Displaying user password value from cookie
          out.print("</br>");
    out.print("Password is: "+c[2].getValue());

    out.close();
  }
        catch(Exception e)
        {
    System.out.println(e);
   }
 }
}
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
 id="WebApp_ID" version="4.0">

 <display-name>Cookie Example</display-name>
 <welcome-file-list>
 <welcome-file>index.jsp</welcome-file>
 </welcome-file-list>
<servlet>
 <servlet-name>Servlet1</servlet-name>
 <servlet-class>ServletTest</servlet-class>
```

```
</servlet>
<servlet-mapping>
 <servlet-name>Servlet1</servlet-name>
 <url-pattern>/login</url-pattern>
</servlet-mapping>
<servlet>
 <servlet-name>Servlet2</servlet-name>
 <servlet-class>ServletTest1</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>Servlet2</servlet-name>
 <url-pattern>/welcome</url-pattern>
</servlet-mapping>
</web-app>
```

Output:

Figure 1: Welcome page

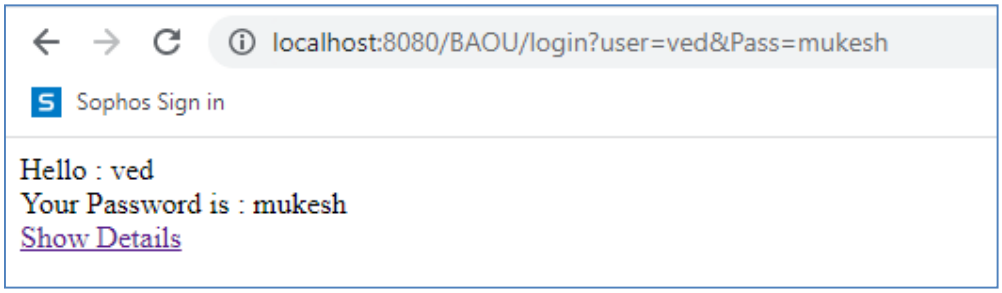


Figure 2: After Clicking Login

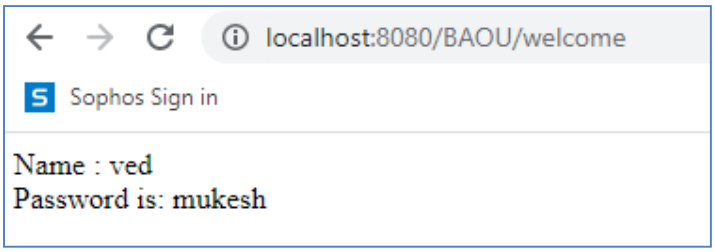


Figure 3: After Clicking Show Details

Check your progress 3

1. SessionIDs are stored in cookies.

a. True

b. False

c. May be

d. Can't say

2. What is the maximum size of cookie?

a. 40 KB

b. 4 MB

c. 4 bytes

d. 4 KB

4.5 URL Rewriting

URL Rewriting is an approach in which a session (unique) identifier gets appended with each request URL so that the server can identify the user session.

For example if we apply URL rewriting on,

- http://localhost:8080/BAOU/ServletTest, it will become something like
- http://localhost:8080/ BAOU/ServletTest?jSessionId=hello

where jSessionId=hello is the attached session identifier and value hello will be used by server to identify the user session.

Query string is a name-value pair separated using an equal = sign, a name-value pair is separated from another name-value pair using the ampersand (&) sign. Query string always should start from Question mark (?). In a Servlet code, we have to use getParameter() and getParameterNames() methods to retrieve a name value pair.

There are several advantages of URL rewriting like,

- It is browser independent and even if user's browser does not support cookie or in case user has disabled cookies, this approach will work.
- Here, we need not required to submit extra hidden parameter.

At the same time, this approach has some disadvantages like we need to regenerate every url to append session identifier and this need to keep track of this identifier until the process gets completed.

Example: In the following example we are storing, forwarding and retrieving user data through URL Rewriting to manage session.

index.jsp

```
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  <title>Servlet Example</title>
 </head>
 <body>
  <h1>Welcome to BAOU</h1>
        <h3>URL Rewriting Example</h3>
  <form method="post" action="validate">
  Name:<input type="text" name="user" /><br/>
  Password:<input type="text" name="pass" ><br/>
  <input type="submit" value="submit">
</form>



 </body>
</html>
```

ServletTest.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletTest extends HttpServlet
{
        public void doPost(HttpServletRequest req, HttpServletResponse res)
          {
    try{
       String name = req.getParameter("user");
      String pass = req.getParameter("pass");
      if(pass.equals("ved"))
      {
        res.sendRedirect("Second?user_name="+ name);
      }
  }
  catch(Exception e)
  {
     System.out.println(e);
   }
 }

}
```

ServletTest1.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletTest1 extends HttpServlet
{
 public void doGet(HttpServletRequest req, HttpServletResponse res)
 {
   try{
      res.setContentType("text/html");
```

```
                PrintWriter out = res.getWriter();
                String user = req.getParameter("user_name");
      out.println("Welcome "+user);
   }
        catch(Exception e)
        {
     System.out.println(e);
    }
 }
}
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
 id="WebApp_ID" version="4.0">

 <display-name>URL Rewriting Example</display-name>
 <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
 </welcome-file-list>

        <servlet>
     <servlet-name>First</servlet-name>
     <servlet-class>ServletTest</servlet-class>
   </servlet>
   <servlet-mapping>
     <servlet-name>First</servlet-name>
     <url-pattern>/validate</url-pattern>
   </servlet-mapping>

   <servlet>
```

```
    <servlet-name>Second</servlet-name>
    <servlet-class>ServletTest1</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Second</servlet-name>
    <url-pattern>/Second</url-pattern>
  </servlet-mapping>
</web-app>
```

Output:

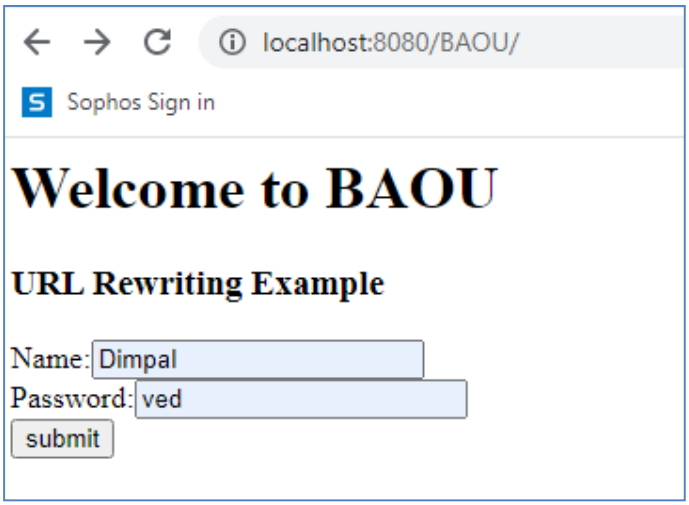


Figure 1: Welcome page

After Clicking submit button, request goes to first servlet, which passes the user data by invoking second servlet through sendRedirect method of HttpResponse and displays user data as shown below**:**

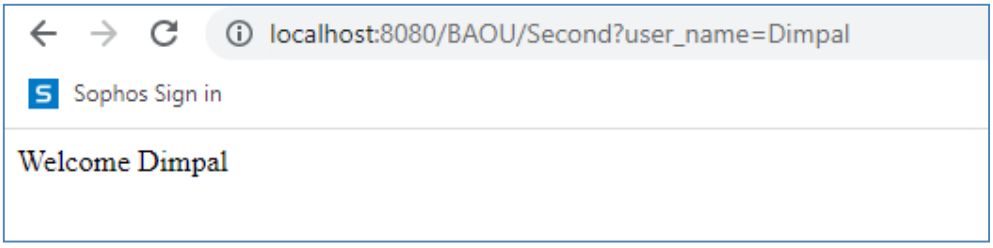


Figure 2: Final output with URL Rewriting

Check your progress 4

Which of the following leads to high network traffic?

a. URL rewriting

b. Hidden form fields

c. SSL sessions

d. Cookies

4.6 Hidden Form Field

In Hidden Form Field, user has to write the response such a way that user has to move values explicitly between client and server. In Hidden Form Field a hidden (invisible) text field is used for maintaining the state of the user. We store the information in the hidden field and get it from another servlet code. This approach is better if we have to submit form in all the web pages and we do not want to depend on the browser. For example

- <input type=”hidden” name=”sessionId” value=”unique value”/>

is a hidden form field which will not be displayed to the user but its value will be send to the server and can be retrieved using request.getParameter(“sessionId”) in servlet code.

We cannot use this approach for static pages like HTML hence HTML pages cannot participate in session tracking.

Example: In the following example we are retrieving user data from user in index.jsp and then passes user data in the first servlet (**ServletTest)**. In the first servlet (**ServletTest)** we again retrieves user data passed from index.jsp and then passes the same user data in second servlet **(ServletTest1)** using hidden form field. In the second servlet **(ServletTest1)** we retrieves it and prints it.

index.jsp

```
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  <title>Servlet Example</title>
 </head>
 <body>
  <h1>Welcome to BAOU</h1>
        <h3>Hidden Form Field Example</h3>
 <form method="post" action="First">
                Name:<input type="text" name="user" /><br/>
                City:<input type="text" name="pass" ><br/>
 <input type="submit" value="submit">
```

```
</form>
 </body>
</html>
```

ServletTest.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletTest extends HttpServlet
{
        public void doPost(HttpServletRequest req, HttpServletResponse res)
          {
    try{
                    res.setContentType("text/html");
                    PrintWriter out = res.getWriter();
                    String user = req.getParameter("user");
                    out.println("<h1>Welcome to BAOU</h1>");
                    out.println("<h3>Please click on Submit Button </h3>");
                    out.println("<form action='Second'>");
                    out.println("<input type='hidden' name='user' value='"+user+"'>");
                    out.println("<input type='submit' value='submit' >");
                    out.println("</form>");
  }
  catch(Exception e)
  {
     System.out.println(e);
   }
 }

}
```

ServletTest1.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
public class ServletTest1 extends HttpServlet
{
 public void doGet(HttpServletRequest req, HttpServletResponse res)
 {
   try{
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      String user = req.getParameter("user");
      out.println("Welcome: "+user);
      out.close();
   }
         catch(Exception e)
         {
     System.out.println(e);
    }
 }
}
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
 id="WebApp_ID" version="4.0">

 <display-name>Hidden Form Field Example</display-name>
 <welcome-file-list>
         <welcome-file>index.jsp</welcome-file>
 </welcome-file-list>

         <servlet>
      <servlet-name>First</servlet-name>
      <servlet-class>ServletTest</servlet-class>
```

```
  </servlet>
  <servlet-mapping>
     <servlet-name>First</servlet-name>
     <url-pattern>/First</url-pattern>
  </servlet-mapping>

  <servlet>
     <servlet-name>Second</servlet-name>
     <servlet-class>ServletTest1</servlet-class>
  </servlet>
  <servlet-mapping>
     <servlet-name>Second</servlet-name>
     <url-pattern>/Second</url-pattern>
  </servlet-mapping>
</web-app>
```

Output:

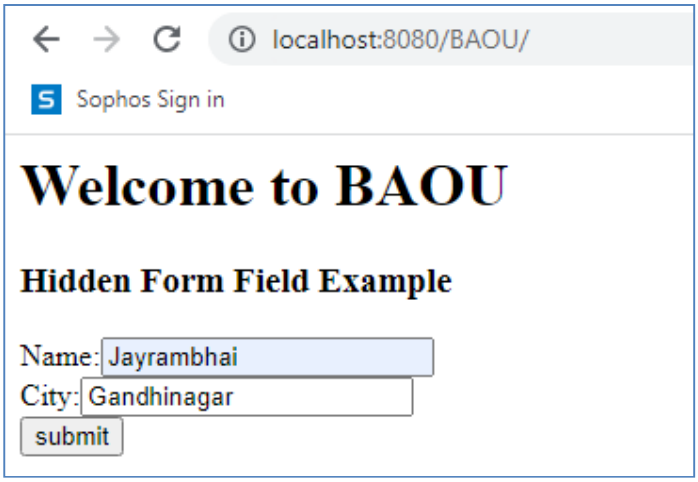


Figure 1: Welcome page

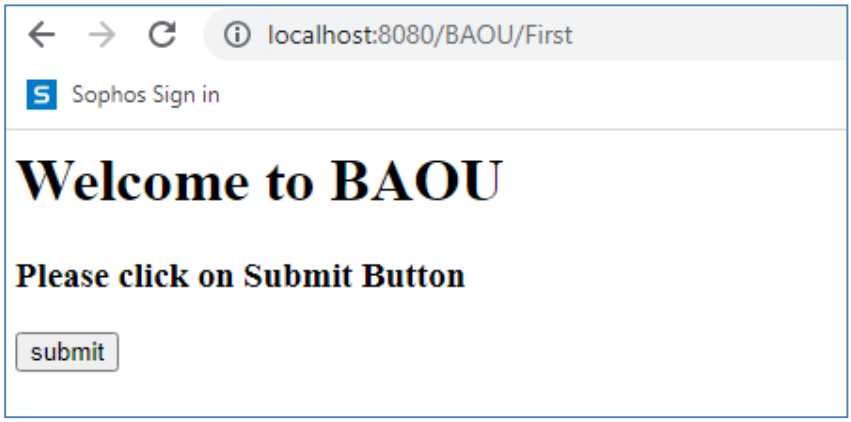


Figure 2: First Servlet with Submit Button

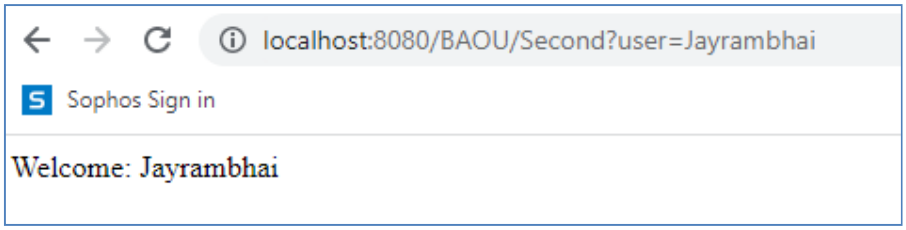


Figure 3: Second Servlet prints user data

Check your progress 5

Which method creates unique fields in the HTML which are not shown to the user?

a. User authentication

b. URL writing

c. HTML Hidden field

d. HTML invisible field

4.7 Http Session

Servlets provide a convenient session-tracking mechanism using the HttpSession API. Session tracking in servlet using HttpSession is very simple and it has following steps:

Get the associated session object (HttpSession) using request.getSession(). Then to get the particular value out of session object, use getAttribute(String) method on the HttpSession object. To store any data in a session use setAttribute(key,object) method on a session object. To remove the session data use removeAttribute(key) method to discard a object with a given

key. To invalidate the session use invalidate() method on session object. This method is used to logout the logged in user.

Example: In the following example we are storing and retrieving users data through HttpSession in servlet1 and servlet2.

index.jsp

```
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  <title>Servlet Example</title>
 </head>
 <body>
  <h1>Welcome to BAOU</h1>
        <h1> HttpSession Example</h1>
        <form action="login">
                User Name:<input type="text" name="user"/></br>
                Password:    <input type="password" name="Pass"/></br>
                <input type="submit" value="Login"/>
        </form>
 </body>
</html>
```

ServletTest.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletTest extends HttpServlet
{
        public void doGet(HttpServletRequest req, HttpServletResponse res)
          {
    try{
       res.setContentType("text/html");
       PrintWriter out = res.getWriter();

       String name = req.getParameter("user");
```

```
        String password = req.getParameter("Pass");
        out.print("Hello : "+name);
                    out.print("</br>");
        out.print("Your Password is : "+password);

        //Creating Session
        HttpSession ses=req.getSession();
                    ses.setAttribute("uname",name);
                    ses.setAttribute("upass",password);
        out.print("<br><a href='welcome'>Show Details</a>");
        out.close();
  }
  catch(Exception e)
  {
     System.out.println(e);
   }
 }
 }
```

ServletTest1.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletTest1 extends HttpServlet
{
 public void doGet(HttpServletRequest req, HttpServletResponse res)
 {
   try{
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
       HttpSession ses=req.getSession(false);
   String Name=(String)ses.getAttribute("uname");
      String Pass=(String)ses.getAttribute("upass");
      out.print("Name is: "+Name + "</br>");
```

```
    out.print(" Password is: "+Pass);
    out.close();
  }
        catch(Exception e)
        {
    System.out.println(e);
   }
 }
}
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
 id="WebApp_ID" version="4.0">

 <display-name>HttpSession Example</display-name>
 <welcome-file-list>
 <welcome-file>index.jsp</welcome-file>
 </welcome-file-list>
<servlet>
 <servlet-name>Servlet1</servlet-name>
 <servlet-class>ServletTest</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>Servlet1</servlet-name>
 <url-pattern>/login</url-pattern>
</servlet-mapping>
<servlet>
 <servlet-name>Servlet2</servlet-name>
 <servlet-class>ServletTest1</servlet-class>
</servlet>
```

```
<servlet-mapping>
 <servlet-name>Servlet2</servlet-name>
 <url-pattern>/welcome</url-pattern>
</servlet-mapping>
</web-app>
```

Output:

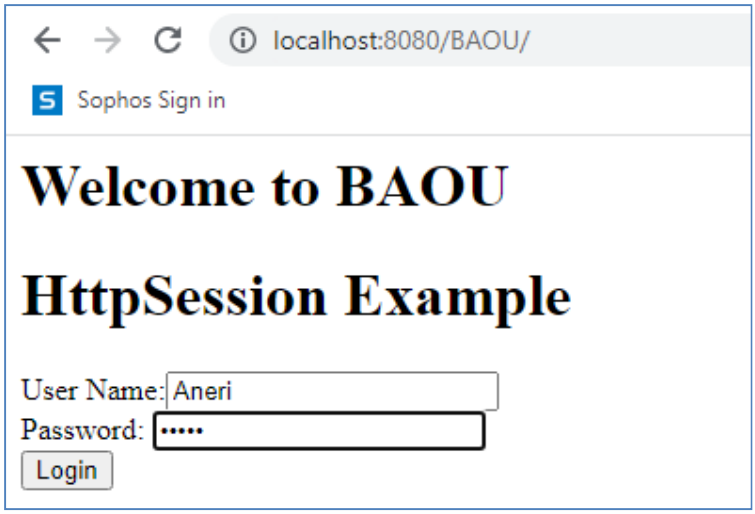


Figure 1: Welcome page

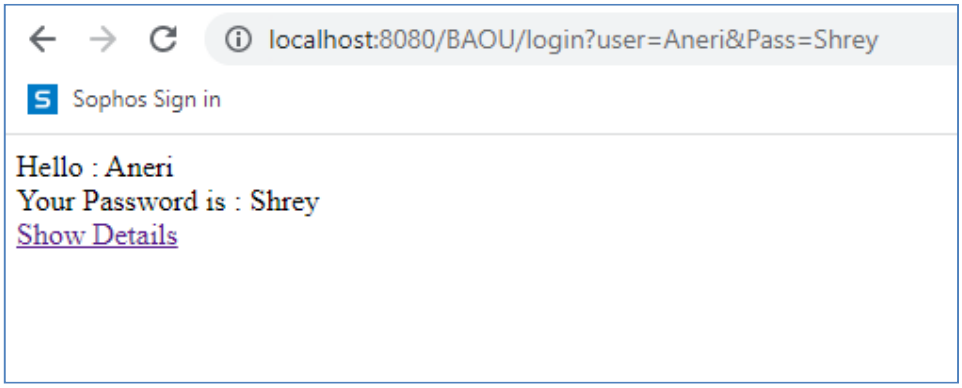


Figure 2: Servlet1 with user and password

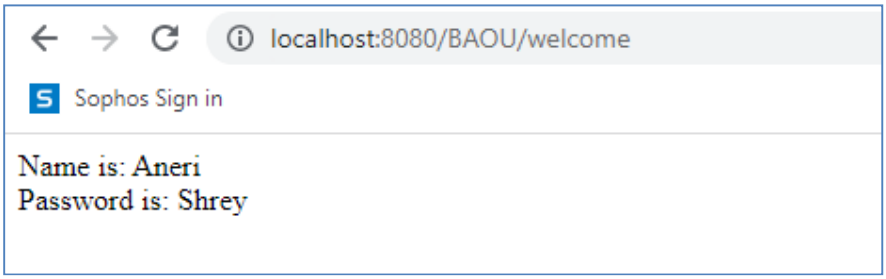


Figure 3: Servlet2 displays user and password

Check your progress 6

How you can destroy the session in Servlet?

a. kill()

b. invalidate()

c. destroy()

d. None of these

4.8 Let Us Sum Up

In this unit we have learnt how each client requests and communication can be stored and retrieved later using session tracking mechanism. HTTP is a stateless protocol. All requests and responses are independent. Sometimes we need to manage and store client's activity across multiple requests. This is possible by creating a session. Session Management is a mechanism used by the Web container to store session data for a particular user. Here we have discussed four different techniques used by Servlet application for session management like Cookies, Hidden form field, URL Rewriting and HttpSession. Each one is best in particular situation and having their pros and cons.

4.9 Answer for Check Your Progress

Check your progress 1: a
Check your progress 2: 1. b 2. d
Check your progress 3: 1. a 2. d
Check your progress 4: a
Check your progress 5: c
Check your progress 6: b

4.10 Glossary

1. Stateless: A stateless refers each communication as a separate event, unrelated to other communications of the same type.
2. Stateful: A stateful on the other hand refers each communication as a part of a broader sequence and responds differently to the same inputs based on the history.

4.11 Assignment

1. Define Session management and its importance.
2. Discuss Cookie with its session mechanism process.
3. Discuss advantages and disadvantages of URL rewriting and hidden form field.
4. Differentiate between Cookie and HttpSession.

4.12 Activities

- Develop a small e-commerce application to maintain user's activity.

4.13 Case Study

- Explore different web application (especially e-commerce related) and analyse security implementation in those application.

4.14 Further Readings

- https://docs.oracle.com/javaee/5/tutorial/doc/bnafe.html
- https://www.w3schools.blog/hidden-field-in-servlet
- https://www.digitalocean.com/community/tutorials/java-session-management-servlet-httpsession-url-rewriting

BLOCK 4: JSP, Expression Language and JSTL

Block Introduction

In this block, the student will learn and understand about the basics of JSP. JSP is a server side technology used to create a dynamic webpage using java. It can be considered an extension of Servlet API. Both HTML and JSP tags are present in Java Server Pages. JSP provides various implicit objects allowing developer to use different directives and action elements to make the application more productive.

We have also discussed JSP Expression Language. It is mainly used to eliminate java code from the JSP. This expression statements increase JSP code readability and reusability. At last, JSTL is discussed. It is a library containing several tags that will remove JSP scriptlet code from a JSP page by providing already implemented common functionalities. It supports common, structural tasks like iteration and conditions, tags for manipulating XML documents, internationalization tags and SQL tags.

Block Objective

After learning this block, you will be able to:

- Define JSP Architecture and Life Cycle
- Use JSP Declaration, Scriptlet and Expression Tags
- Use JSP Implicit Objects
- Use JSP Directives
- Use JSP Action Elements
- Use Different EL Implicit Objects
- Use EL Operators
- Use JSTL Core, Function, Formatting, XML and SQL Tags
- Define Servlet and JSP Relationship

Block Structure

Unit 1: basics of JSP

Unit 2: JSP Objects and Directives

Unit 3: JSP Expression Language (EL)

Unit 4: JSTL

Block Summary

JavaServer Pages helps in developing web pages having include dynamic content. In this block we have discussed the importance of JSP in web application. Various building blocks to write a JSP page were discussed. JSP allows the developer to insert Java code into the HTML pages by using special JSP tags. It can be either HTML or XML with JSP actions and commands. Then various Implicit Objects were also discussed. We have also discussed JSP EL which provides a platform to easily access application data stored in JavaBeans components. JSP EL allows us to create expressions both arithmetic and logical. At last we have also focused on JSTL. JSTL allows us to program our JSP pages using various tags, rather than the scriptlet code. With JSTL we can do nearly everything that regular JSP scriptlet code can do, it leads to faster development of JSP pages, code re-usability.

Block Assignment

Short Answer Questions:

1. How does JSP Initialization take place?
2. What is the JSP Scriptlet?
3. What are some of the advantages of using JSP?
4. List some important JSP Action Tags.
5. What is JSP Expression Language (EL)?
6. What is JSTL?

Long Answer Questions:

1. Discuss different Life-Cycle methods of JSP.
2. Explain various scope values for <jsp.useBean> tag.
3. Discuss various Implicit Objects of JSP.
4. Discuss various Implicit Objects used in the Expression Language.
5. Explain various JSTL Core tags.

UNIT 1: BASICS OF JSP

Unit Structure

1.0 Learning Objectives

After learning this Unit, you will be:

- Define JSP Architecture
- Write JSP Syntax
- Define JSP Life Cycle
- Write JSP Scriptlet Tag
- Write JSP Expression Tag
- Write JSP Declaration Tag

1.1 Introduction

In the early days of Servlets, Servlet was not that much popular among ASP programmers as ASP supports tag-based programming that servlet doesn't support. For any programmer to work with Servlet strong knowledge of java is required. It is not an easy task for an ASP programmer to learn Java. Servlet suffers from following limitations:

- It is not preferable for non-java programmers as strong knowledge of Java is required to work with Servlets.
- To write a HTML code in the Servlet program is a complex and error-prone process.
- Programmer mixes up presentation logic (HTML) and business logic (Java code).
- Any changes made in the Servlet file will be reflected only after recompilation of the Servlet file leading to overloading of Web Application.
- Doesn't provide implicit object. So, you need to write additional code to access those objects.
- ServletConfig in web.xml is mandatory.

To solve these problems Sun Microsystems has given a tag-based technology called JSP having all the features of Servlet. So, any programmer can use JSP without having strong knowledge of Java or Servlet. JSP is a server side technology used to create a dynamic webpage using java as the programming language. It is a specification from Sun Microsystems. It can be considered an extension of Servlet API because it offers more functionality than servlet. The JSP pages are simpler to manage than Servlet as we can

differentiate design and development. Both HTML and JSP tags are present in Java Server Pages.

JSP technology was developed to simplify the process of creating pages by separating web presentations from business logic. JSP is a collection of HTML tags and JSP tags along with Java code. Every JSP page can be placed in the root folder or WEB-INF directory of web application. The JSP placed in the root folder is a public resource and this resource can access by the client directly. The JSP placed inside WEB-INF directory is a private resource, which can't access directly by the client. Client can only access this resource using public URL defined inside the web.xml file.

JSP pages are opposite of Servlets. In Servlet we can add add HTML code inside Java code, while in JSP we can add Java code inside HTML using JSP tags. In JSP, we can easily separate Presentation and Business logic. Web page designer can design and update JSP pages by creating the presentation layer and java programmer can write server side complex code independently without concerning the web design. Both the Presentation and Business layers will easily communicate over HTTP requests. The file extension of the source file of a JSP page will be .jsp. The extension of the source file of a JSP page fragment will be .jspf.

1.2 JSP Life Cycle

JSP provides tag-based programming so it is very easy to learn and code. JSP life cycle can be defined as the process from its creation till the destruction which is somewhat similar to a Servlet life cycle except an additional step required to compile a JSP into Servlet. Generally life cycle of a JSP page consists of 6 stages as discussed below.

1. Page Translation

When a browser sends an HTTP request to the web server the web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. Then the JSP engine loads the JSP page from disk and translates it into a Servlet form. In this translation all template text is translated to println( ) statements and all JSP elements are translated to Java code. In this phase, the JSP container validates the correctness of the JSP pages and tag files.

Example: Let's check what happens when a code of Test.jsp file is translated into Servlet. The code inside the <% %> is JSP code.

```
<html>
  <head>
    <title>Welcome to Babasaheb Ambedkar Open University</title>
  </head>
```

```
  <%
     float PI = 3.14;
  %>
  <body>
     The Value of PI is:
     <% out.println(PI); %>
  </body>
</html>
```

The above JSP page (Test.jsp) will be translated in to following Servlet file.

```
public class Test_jsp extends HttpServlet
{
 public void _jspService(HttpServletRequest request, HttpServletResponse response)
                        throws IOException,ServletException
  {
    PrintWriter out = response.getWriter();
    response.setContenType("text/html");
    out.write("<html><body>");
    float PI = 3.14;
    out.write("The Value of PI is:");
    out.print(PI);
    out.write("</body></html>");
  }
}
```

As it is done automatically by the web container you are not required to worry about how a JSP page will be converted to a Servlet.

2. Page Compilation

The JSP engine compiles the created Servlet file into a java Servlet class file.

3. Class Loading

The java Servlet class file that was compiled from the JSP source will be loaded into the container.

4. Initialization

When a container loads a JSP, it invokes the jspInit() method before servicing any requests. If we need to perform JSP-specific initialization we have to override the jspInit() method as shown below:

```
public void jspInit()
{ // Initialization code... }
```

Typically initialization is performed only once.

5. Execution

Whenever a browser requests a JSP file and the page has been loaded and initialized, the JSP engine invokes the _jspService() method in the JSP. The _jspService() method takes an HttpServletRequest and an HttpServletResponse objects as its arguments as shown below:

```
void _jspService(HttpServletRequest request, HttpServletResponse response)
{ // Service handling code... }
```

The _jspService() method is invoked once per request and is responsible for generating the response for that request.

6. jspDestroy()

The jspDestroy() method is equivalent to the destroy method for Servlets. We have to override jspDestroy method whenever we need to perform any cleanup, such as releasing database connections or closing open files. The jspDestroy() method has the following signature:

```
public void jspDestroy()
{ // Your cleanup code goes here. }
```

Check your progress 1

1. Which technology do we mix our business logic with the presentation logic?

a. HTML

b. JSP

c. Servlet

d. None of these

2. JSPs eventually are compiled into Java Servlets. You can do as much with JSPs as you can do with Java Servlets.

a. True

b. False

3. A JSP page consists of which tags?

a. HTML tags

b. JSP tags

c. Both a & b

d. None of these

4. For What purpose JSP is used?

a. Server-side dynamic content generation

b. Client Side language for validation

c. Web page designing

d. None of these

1.3 JSP Architecture

The architecture of JSP is generally based on the Model-View-Controller (MVC) pattern. It separates the page's business logic from its presentation. This allows the programmer to easily change the page's look and feel without modifying the underlying business logic.

Generally the JSP architecture consists of three main components:

1. The client requests for the JSP page.
2. The JSP engine receives request and processes the JSP page and generates a response.
3. The JSP container manages the lifecycle of JSP pages.

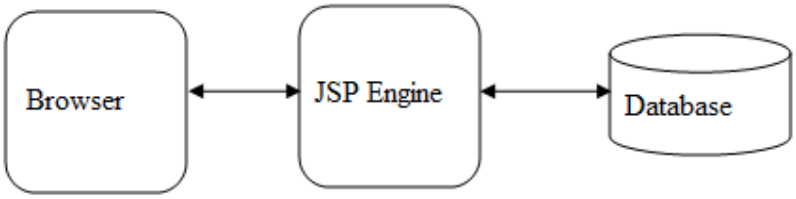


Figure 1: JSP Architecture

JSP pages are typically used to display data from a database. The JSP engine reads the JSP and translates it into a Servlet. The Servlet then manipulates the database and retrieves the data. The data is then transferred to the JSP page, which displays it to the user.

The Servlet file is cached and reused for subsequent requests when JSP is first accessed. This enables the JSP page to be displayed quickly, without having to access the database each time.

The JSP architecture is flexible enough and can be customized whenever required to meet the needs of our application. For example, JSP pages can used to create RSS feeds or generate PDF files.

Check your progress 2

Which technology do we mix our business logic with the presentation logic?

a. JSP

b. Servlet

c. Both a & b

d. None of these

1.4 JSP Declaration Tag

This tag is often used to provide all the java declarations like variable declarations, method definitions, classes declarations and so on. You can declare a static member, an instance variable and methods inside the declaration tag. You can use declarations to declare one or more variables and methods at the category level of the compiled Servlet. The most significant fact is that they are declared at a class level rather than in the body of the page. The variables and methods can then be employed by Java code within the remainder of the page. While writing a declaration during a JSP pages please remember these rules:

- Must end the declaration with a semicolon.
- <% int v=10;%>
- You can directly use the variables or methods that are declared in packages imported by the page directive, without declaring them in a declaration element.
- You can declare any number of variables or methods within one declaration element, as long as you end each declaration with a semicolon. The declaration must be valid in the Java programming language.

Syntax of declaration tag:

<%! Declaration %>

Example:

```
<%@ page language=”java” contentType=”text/html; charset=ISO-8859-1”
 pageEncoding=”ISO-8859-1”%>
<html>
<head>
<title>Declaration Tag Example</title>
</head>
```

```
<body>
<h3>Welcome to BAOU</h3>
<h3>Use of Declaration Tag in JSP</h3>
<%! int d = 5, v = 8, n = 10, ans=0; %>
<%
        ans = d + v + n;
        out.println("The summation of numbers is:" + ans);
%>
</body>
</html>
```

In above example we have declared four variables inside declaration tag.

Check your progress 3

1. Which of the following scripting elements can be used to declare methods and fields?

a. scriptlet tag

b. expression tag

c. declaration tag

d. All of these

2. In JSP, java code can be written inside the jsp page using _____

a. scriptlet tag

b. expression tag

c. declaration tag

d. JSP include directive

1.5 JSP Scriptlet Tag

Scriptlets are java code enclosed within <% and %> tags. This scripting element is used to provide a block of java code. It can contain any number of statements, variable or method declarations, or expressions that are valid within the page scripting language. JSP container transfers statements in _jspservice() method while generating servlet from jsp. For each

request of the client, service method of the JSP gets called so the code inside the Scriptlet block executes for every request. Within a scriptlet element, you can do the following:

- Declare variables or methods for later use.
- Write valid expressions in the page scripting language.
- Use any implicit objects or any object declared with the element.
- All text, HTML tags, JSP elements must be written outside the scriptlet.

Syntax of Scriptlet tag:

<% java code %>

In below example, Scriptlet tags enclose java code.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<html>
<head>
<title>Scriptlets Example</title>
</head>
<body>
 <h3>Welcome to BAOU</h3>
 <h3>Demonstrating the use of Scriptlets in JSP</h3>
 <%
 int a = 5;
 int b = 7;
 int c = 9;
 out.println("a is: " + a + "<br>" + "b is:" + b + "<br>" + "c is:" + c + "<br>");
 out.println("Multiplication of numbers is: " + a * b * c + "<br>");
 out.println("Addition of numbers is:" + (a + b + c));
 %>
</body>
</html>
```

Check your progress 4

The code placed in scriptlet goes to _____ method.

a. _jspDestroy()

b. _jspInit()

c. _jspService(_,_)

d. None of these

1.6 JSP Expression Tag

This scripting element is often used to evaluate only java expression and display that expression value onto the client browser. The expression element contains a Java expression that returns a value. This value is then written back to the HTML page. The Expression tag can contain any kind of expression that is valid and consistent with the Java Language Specification. This includes variables, method calls or any object that contains a toString() method. It evaluates the given expression and displays generated results on the browser windows. It allows you to create expressions like arithmetic and logical. It facilitates produces scriptless JSP pages. Simply, anything that returns a result is called an expression.

Syntax of expression tag:

```
<%= expression %>
```

Example:

```
<%@ page language=”java” contentType=”text/html; charset=ISO-8859-1”
 pageEncoding=”ISO-8859-1”%>
<html>
<body>
 Current Time is:
      <%=java.util.Calendar.getInstance().getTime()%>
</body>
</html>
```

Above example displays current time using expression.

Check your progress 5

JSP Expression tag is used for displaying output on browser.

a. True

b. False

The following example demonstrates the use of all the JSP page building blocks in single JSP file named buildBlock.jsp.

```
<%@ page language=”java” contentType=”text/html; charset=ISO-8859-1”
```

```
   pageEncoding=”ISO-8859-1”%>
<!DOCTYPE html PUBLIC “-//W3C//DTD HTML 4.01 Transitional//EN”
“http://www.w3.org/TR/html4/loose.dtd”>
<html>
<head>
<meta http-equiv=”Content-Type” content=”text/html; charset=ISO-8859-1”>
<title> Welcome to Babasaheb Ambedkar Open University </title>
</head>
<body>
<h3> JSP example with Declaration, Scriplet, Comments and Expressions <h3>
<%-- This is a comment syntax to be used in JSP page  --%>
<%-- following statements are Scriptlets  --%>
<% out.println(“BAOU is located at Ahmedabad”); %>
<% out.println(“The number is “); %>
<%-- following statement is a declaration block --%>
<%! int value1 = 15; int value2 = 15; %>
<%-- following statements are expression blocks --%>
<%= value1* value2 %>
Today's date: <%= (new java.util.Date()).toLocaleString()%>
</body>
</html>
```

1.7 Let Us Sum Up

In this unit we learnt that JSP is a server side technology which helps the programmer to create a webpage dynamically using java as the programming language.

To understand the JSP, we have first discussed JSP page execution through its life cycle and also explored JSP architecture. We have also discussed the building blocks of a JSP page. The first is declaration section; which allows us to declare variables and methods. A scriptlet section contains any number of java language statements, variable or method declarations, or expressions which is valid in page scripting language. An expression section allows us to send the response back to the browser.

1.8 Answer for Check Your Progress

Progress 1: 1. c 2. a 3. c 4. a

Progress 2: b

Progress 3: 1. c 2. a

Progress 4: c

Progress 5: a

1.9 Glossary

Servlet: A servlet is a Java programming language that is used to design and deploy dynamic web pages using the Java Programming Language to extend the capabilities of servers that host applications accessed by means of a request-response programming model.

JSP: A server technology used to create a dynamic web page in java.

Presentation: A Graphical View presented to the user

JSP Engine: JSP engine is a container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages.

1.10 Assignment

1. Define JSP. Explain the JSP Architecture?
2. List the benefits of JSP.
3. Explain JSP page life cycle in detail.
4. Explain different JSP building blocks.

1.11 Activities

Understand and implement various JSP life cycle stages.

1.12 Further Readings

- https://www.tutorialspoint.com/jsp/jsp_overview.htm
- https://www.edureka.co/blog/jsp-in-java/
- https://www.studytonight.com/jsp/introduction-to-jsp.php
- https://dotnettutorials.net/lesson/jsp-architecture/

UNIT 2: JSP OBJECTS AND DIRECTIVES

Unit Structure

2.0 Learning Objectives

After learning this Unit, you will be able to:

- Use JSP Implicit Objects
- Use JSP Directives
- Use JSP action elements
- Able to write JSP application

2.1 Introduction

JSP allows us to build powerful web application with the help of various implicit objects. It also allows developer to use different directives and action elements to make the application more robust and efficient.

In Java 2 Standard applications, it is a basic requirement to display data on the command prompt. To perform this operation every time programmer has to prepare PrintStream object with command prompt location as destination location:

```
Printstream print = new PrintStream("C:\Program files\…..\cmd.exe");
print.println("Welcome to BAOU");
```

In core Java applications, the PrintStream object is a routine requirement so that java technology has provided that PrintStream object as a predefined object in the form of out variable in System class.

```
public static final PrintStream out;
```

Similarly, in web applications, all the web developers may require some kind of objects to display some messages. To get these objects you have to write some lines of java code. To fulfil such objects requirement in web applications, JSP technology has provided them as predefined functionality in the form of Implicit Objects. This functionality reduces the burden of writing code from the developers.

2.2 JSP Implicit Objects

Implicit Objects are a group of Java objects which the JSP Container makes available to programmer on each JSP page. These objects can be accessed as built-in variables through scripting elements. It can also be accessed programmatically by using JavaBeans and Servlets. These objects are created automatically for programmer within the service method.

These objects are parsed by the container and plugged into the generated servlet code. They are only available within the JSP service method and not in any other declaration.

There are total 9 implicit objects available in JSP.

1. Request:

It is an instance of javax.servlet.http.HttpServletRequest object. This implicit object is used to process the request sent by the client.

Example:

index.html

```
<!DOCTYPE HTML><html lang="en">
<head>
<meta charset="UTF-8">
        <title>Welcome to BAOU</title>
</head>
<body bgcolor="cyan">
 <form action="user.jsp">
  <input type="text" name="uname">
  <input type="submit" value="Click">
  <br/>
 </form>
</body>
</html>
```

user.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Use of implicit request object</title>
</head>
<body bgcolor="cyan">
 <%
  String name = request.getParameter("uname");
```

```
out.print("Welcome" + "  " + name + " "+ "to BAOU");
%>
</body>
</html>
```

In the above example, we are printing the name of the user with a welcome message. Here we are receiving the input from the user on the index.html page and displaying it in the user.jsp page using implicit request object.

2. Response:

This object is the HttpServletResponse object associated with the response to the client. This implicit object is used to process the request and send the response back to the client. This object is by default available to scriptlets and expressions to get response-related data and to set response-related information. If a JSP page wants to redirect a request from one server to another server then it can use the response object and its sendRedirect method.

Example:

index.html

```
<!DOCTYPE HTML><html lang="en">
<head>
<meta charset="UTF-8">
        <title>Welcome to BAOU</title>
</head>
<body bgcolor="cyan">
 <form action="user.jsp">
  <input type="text" name="uname">
  <input type="submit" value="Click">
  <br/>
 </form>
</body>
</html>
```

user.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
```

```
<html>
<head>
<meta charset="ISO-8859-1">
<title> Use of implicit response object </title>
</head>
<body bgcolor="cyan">
 <%
  response.sendRedirect("https://baou.edu.in/");
 %>
</body>
</html>
```

In the above example, we are redirecting the response to BAOU website. Here we are receiving the input from the user on the index.html page and redirecting it on the user.jsp page using implicit response object.

3. PageContext:

The PageContext object encapsulates the environment of a single request for the current JSP page. It is used for accessing page, request, application and session attributes. It belongs to package java.servlet.jsp.PageContext. In JSP applications, with the help of pageContext object we can perform some operations with the attributes in the JSP scopes page, request, session and application like adding an attribute, removing an attribute, getting an attribute and finding an attribute.

Example:

index.html

```
<!DOCTYPE HTML><html lang="en">
<head>
<meta charset="UTF-8">
        <title>Welcome to BAOU</title>
</head>
<body bgcolor="cyan">
 <form action="user.jsp">
  <input type="text" name="uname">
  <input type="submit" value="Click">
```

```
 <br/>
</form>
</body>
</html>
```

user.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Use of implicit pageContext object</title>
</head>
<body bgcolor="cyan">
<%
 String name = request.getParameter("uname");
out.print("Welcome" + "  " + name + " "+ "to BAOU");
pageContext.setAttribute("User", name, PageContext.SESSION_SCOPE);
%>
 <a href="pagecontext.jsp">pageContext User Parameter Access</a>
</body>
</html>
```

pagecontext.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Use of implicit pageContext object</title>
</head>
<body bgcolor="cyan">
<%
```

```
 String name = (String) pageContext.getAttribute("User", PageContext.SESSION_SCOPE);
out.print("Welcome " + name);
 %>
</body>
</html>
```

In the above example, in index.html we are taking user input and storing user's data using Pagecontext with the session scope in welcome.jsp page. With this we can access user details till the user's session is active. Then we are retrieving the stored attributes using the getAttribute method of Pagecontext object in the pagecontext.jsp page.

4. Session:

HttpSession object is associated with the request. A session is an implicit object which is created for HttpSession class to store and access data. By default, session reference is available to scriptlet and expression tags. These tags can access session related data and they can manage session scope attributes. By using session reference these tags can delete session objects by calling session.invalidate() and these reference tags can set session timeout.

Example:

index.html

```
<!DOCTYPE HTML><html lang="en">
<head>
<meta charset="UTF-8">
        <title>Welcome to BAOU</title>
</head>
<body bgcolor="cyan">
 <form action="user.jsp">
  <input type="text" name="uname">
  <input type="submit" value="Click">
  <br/>
 </form>
</body>
</html>
```

user.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

```
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Use of implicit Session object</title>
</head>
<body bgcolor="cyan">
 <%
  String name = request.getParameter("uname");
 out.print("Welcome" + "  " + name + " "+ "to BAOU");
 session.setAttribute("User", name);
 %>
  <a href="sessiondata.jsp">Access Session data</a>
 </body>
</html>
```

sessiondata.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Use of implicit Session object</title>
</head>
<body bgcolor="cyan">
 <%
String name = (String) session.getAttribute("User");
 out.print("Welcome " + name);
 %>
</body>
</html>
```

In the above example, the index.html page displays a text box along with a submit button which would transfer the control to user.jsp page which will display the text user has entered.

It then stores the same variable in the session object so that it can be retrieved on any JSP page until the session becomes inactive. Later on in sessiondata.jsp we are retrieving the variable's value from the session object and displaying it.

5. Application:

This is the ServletContext object associated with the application context. It is an implicit object created for ServletContext class and used to access the data of the ServletContext object. This is used for getting application-wide initialization parameters and to maintain useful data across whole JSP application. The ServletContext object for the web application belongs to the package Javax.servlet.ServletContext.

Example:

index.html

```
<!DOCTYPE HTML><html lang="en">
<head>
<meta charset="UTF-8">
        <title>Welcome to BAOU</title>
</head>
<body bgcolor="cyan">
 <form action="user.jsp">
  <input type="text" name="uname">
  <input type="submit" value="Click">
  <br/>
 </form>
</body>
</html>
```

user.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Use of implicit Application object</title>
</head>
<body bgcolor="cyan">
```

```
<%
 String name = request.getParameter("uname");
out.print("Welcome" + "  " + name + " "+ "to BAOU" +"</br>");
String unvname = application.getInitParameter("University");
out.print("University name is: " + unvname);
%>
</body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
 id="WebApp_ID" version="4.0">
 <servlet>
  <servlet-name>welcome</servlet-name>
  <jsp-file>/user.jsp</jsp-file>
 </servlet>
 <servlet-mapping>
  <servlet-name>welcome</servlet-name>
  <url-pattern>/welcome</url-pattern>
 </servlet-mapping>
 <context-param>
  <param-name>University</param-name>
  <param-value>Babasaheb Ambedkar Open University</param-value>
 </context-param>
</web-app>
```

In the above example, we are using the application object in the user.jsp file to get the initialization parameter “University” from the web.xml configuration file.

6. Config:

This is the ServletConfig object associated with the servlet for current JSP page. This is mainly used for accessing configuration information such as servlet context, servlet name, configuration parameters etc. It belongs to package Javax.servlet.ServletConfig.

Example:

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Use of implicit config object</title>
</head>
<body bgcolor="cyan">
 <%
 String sname = config.getServletName();
 out.print("Servlet Name is: " + sname);
 %>
 </body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
 id="WebApp_ID" version="4.0">
<servlet>
  <servlet-name>HelloServlet</servlet-name>
  <jsp-file>/index.jsp</jsp-file>
 </servlet>
 <servlet-mapping>
```

```
 <servlet-name>HelloServlet</servlet-name>
 <url-pattern>/index</url-pattern>
</servlet-mapping>
</web-app>
```

In the above example, we are calling the getServletName() method of implicit config object for retrieving the servlet name from web.xml file.

7. Out:

This is the PrintWriter object used to send output to the client. The reference of out points JSPWriter subclass object. With the help of out object, we can print HTML tags and plain text data on the browser. It belongs to package javax.servlet.jsp.jspwriter.

Example:

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Use of implicit Out object</title>
</head>
<body bgcolor="cyan">
 <%
 out.print("Today is:" + java.util.Calendar.getInstance().getTime()); %>
 </body>
</html>
```

In the above example, we are using the print method of OUT object for displaying messages to the client.

8. Page:

The page variable is equivalent to this variable of Java programming language. It is used to call the methods defined by the translated servlet class. The reference of page object points current JSP page object.

Example:

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Use of implicit Page object</title>
</head>
<body bgcolor="cyan">
 <%
String pageName = page.toString();
out.println("Page Name is:" + pageName);
%>
</body>
</html>
```

9. Exception:

The exception object represents the Throwable object that was thrown by some other JSP page.

Example:

index.html

```
<!DOCTYPE HTML><html lang="en">
<head>
<meta charset="UTF-8">
        <title>Welcome to BAOU</title>
</head>
<body bgcolor="cyan">
 <form action="divide.jsp">
  Enter First Value:<input type="text" name="first" />
  Enter Second Value:<input type="text" name="second" />
  <input type="submit" value="Click" />
```

```
 <br/>
</form>
</body>
</html>
```

divide.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Welcome to BAOU</title>
</head>
<body bgcolor="cyan">
 <%@ page errorPage="exception.jsp"%>
 <%
  int num1 = Integer.parseInt(request.getParameter("first"));
 int num2 = Integer.parseInt(request.getParameter("second"));
 int result = num1 / num2;
 out.print("Answer is: " + result);
 %>
</body>
</html>
```

exception.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Welcome to BAOU</title>
</head>
<body bgcolor="cyan">
```

```
<%@ page isErrorPage="true"%>
Raised the Exception:
<%=exception%>
</br>
Please varify the entered data.
</body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
 id="WebApp_ID" version="4.0">
 <servlet>
  <servlet-name>welcome</servlet-name>
  <jsp-file>/index.html</jsp-file>
 </servlet>
 <servlet-mapping>
  <servlet-name>welcome</servlet-name>
  <url-pattern>/welcome</url-pattern>
 </servlet-mapping>
</web-app>
```

Check your progress 1

1. The _____ object is created by the web container for each jsp page.
   a. application
   b. config
   c. exception
   d. All of these
2. Which of the following is not implicit object in jsp?
   a. cookies
   b. session

c. page

d. pageContext

3. This object can be used to get initialization parameter from configuration file (web.xml)

a. config

b. application

c. session

d. request

2.3 JSP Directives

Directives are basically used to configure the code that is generated by the container during a translation phase. JSP directives are JSP components that are used to give the instructions to the JSP compiler. JSP Directives can be used to define present JSP page behaviour to include the target resource content into the current JSP page. All the JSP directives are going to be resolved at the time of translating the JSP page in to the Servlet.

Syntax:

<%@Directive_name[attribute-list]%>

There are three types of Directives in JSP:

1. Page Directives
2. Include Directives
3. Taglib Directives

1. Page Directive

The page directives define the properties for the entire JSP page by using its different attributes and setting values of these attributes as per requirements. It is helpful to provide global information for the JSP page. Basically, Page directives are used to give the instructions to the JSP compiler like which package to import, which language we are writing etc. So, basically Page directives are used for supplying compile-time information to the container for generating a servlet.

Syntax: <%@ page attributeName=”values” %>

Attributes of JSP page directive are listed in following table:

Attribute	**Description**
import	The import attribute is used to import class, interface or all the members of a package. Example: <%@ page import="java.util.Date" %>

info	This attribute gives an instruction to JSP compiler to override getServletInfo() method to return the info about JSP file. Example: <%@ page info=”Welcome to BAOU”%>
ContentType	This attribute is a page directive attribute that gives instructions to the JSP compiler like what type of content we are sending to the client. We can specify the content type like text/xml, text/css, etc. Example: <%@ page contectType=”text/xml”%>
buffer	This attribute is page directive attribute which give an instruction to JSP compiler what is the buffer size can be taken by out implicit variable of JspWriter (by default 8KB). Example: <%@ page buffer=”5kb”%>
autoflush	By default auto flush is true but if we want to control flushing on our own requirement we have to use this autoflush page directive. Example: <%@ page autoFlush=”false”%>
isErrorPage	isErrorPage is a directive attribute that is used to create the error page. Example: <%@ page isErrorPage=”true”%>
errorPage	If our JSP file containing any exception and if we want to display any error page then we can use this errorPage page directive attribute. Example: <%@ page errorPage=”error.jsp”%>
session	This attribute is a page directive attribute using which we can make the session enable or disable for a particular JSP page. By default, the session is true but if we want to make it false or true, we can use this session attribute. Example: <%@ page session=”false”%>
extends	This attribute is a page directive attribute using which we can create our JSP servlet program by extending from any other class. Example: <%@ page extends=”BAOU.Java.ServletDirective”%>

isThreadSafe	By default, every JSP page is true but if we want to make whether it is thread-safe or not, we have to mention isThreadSafe page directive attribute either to be true or false. True indicates multithreading and false indicates that the servlet should implement SingleThreadModel. Example: <%@ page isThreadSafe=”true”%>
pageEncoding	This attribute is a page directive attribute using which we can specify what charset we are using in JSP. Example: <%@ pageEncoding=”UTF-8″%>
isELIgnored	This attribute is a page directive attribute used to define ENUM data type. Its default value is false. Example: <%@ page isELIgnored=”false”%>

Check your progress 2

The pageContext object can be used to set or get or remove attributes from which of the following scopes

a. request

b. session

c. application

d. All of these

2. Include Directive

JSP include directive is used to include one file to the another file. The included file can be HTML, JSP, text files etc. This directive tells the container to merge the content of other external files with the present JSP during the translation phase. Include directives can be used anywhere in your JSP page.

Syntax:

<%@ include file=”—”%>

Example:

Index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
   <%@ include file="header.jsp" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
```

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP Page Directive Usage</title>
</head>
<body>
<a>This is the main JSP file</a>
</body>
</html>
```

header.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

</head>
<body>
<a>Included file in index.jsp : </a>
<%
  for(int i=1;i<5; i++)
  {
                out.println(i);
  }
%>
</body>
</html>
```

In the above example, we used include directive where we are including the file header.jsp into the main file(index.jsp) and gets the output of both main file and included file.

3. Taglib Directive

JSP taglib directive is used to define the tag library with “taglib” as the prefix. The main purpose of Taglib Directives is to make available user-defined tag library into the present JSP pages.

Syntax:

<%@ taglib uri=”—” prefix=”—”%>

Here, “uri” attribute is a unique identifier in tag library descriptor and “prefix” attribute is a tag name.

Example:

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Example of JSP Taglib Directives</title>
</head>
<body>
 <c:out value="${'Welcome to BAOU!!!'}" />
</body>
</html>
```

In the above example “taglib” is defined with attributes uri=”http://java.sun.com/jsp/jstl/core” and prefix="c"%>. Here, “c” is the custom tag defined and it can be used anywhere.

2.4 JSP Action Elements

JSP actions use the construct in XML syntax to control the behavior of the servlet engine. Through JSP actions a file can be added into another page dynamically, a bean component can be reused, a user can be forwarded from one page to another page. Unlike directives, actions are re-evaluated each time the JSP page is accessed. These tags are used to remove or eliminate scriptlet code from our JSP page because scriplet code are technically not recommended nowadays. It's considered to be bad practice to put java code directly inside your JSP page.

JSP standard tags starts with the jsp: prefix. There are many JSP Standard Action tags.

Syntax:

<jsp:action_name attribute=”value” />

The following are some JSP Standard Action Tags available:

Action Tag	Description
jsp:forward	It forwards the request to another page. Syntax of <jsp:forward> : <jsp:forward page="URL of the another static, JSP OR Servlet page" />
jsp:useBean	It instantiates a JavaBean. This action is useful when we want to use Beans in a JSP page, through this tag we can easily invoke a bean. Syntax of <jsp:useBean>: <jsp: useBean id="unique_name_of_bean" class="package_name.class_name" /> Once Bean class is instantiated using above statement, we have to use jsp:setProperty and jsp:getProperty actions to use the bean's parameters.
jsp:getProperty	It is used to retrieve or fetch the value of Bean's property. syntax of <jsp:getProperty>: <jsp: useBean id="unique_name_of_bean" class="package_name.class_name" /> .... <jsp:getProperty name="unique_name_of_bean" property="property_name" />
jsp:setProperty	It store data in property of any JavaBeans instance. This action tag is used to set the property of a Bean, while using this action tag, we may need to specify the Bean's unique name (it is nothing but the id value of useBean action tag). syntax of <jsp:setProperty>: <jsp: useBean id="unique_name_of_bean" class="package_name.class_name" /> .... <jsp:setProperty name="unique_name_of_bean" property="property_name" />
jsp:include	It includes the runtime response of a JSP page into the current page. In <jsp:include> the file is being included during request processing. Syntax of <jsp:include> : <jsp:include page="page URL" flush="Boolean Value" />

jsp:plugin	It generates client browser-specific construct that makes an OBJECT or EMBED tag for the Java Applets. It is used to introduce Java components into jsp, i.e., the java components can be either an applet or bean. It detects the browser and adds <object> or <embed> tags into the file Syntax: <jsp:plugin type="applet/bean" code="objectcode" codebase="objectcodebase">
jsp:fallback	It supplies alternate text if java plugin is unavailable on the client. You can print a message using this, if the included jsp plugin is not loaded.
jsp:element	Defines XML elements dynamically
jsp:attribute	It defines dynamically defined XML element's attribute. This tag is used to define the XML dynamically i.e. the elements can be generated during request time than compilation time It actually defines the attribute of XML which will be generated dynamically. Syntax: <jsp:attribute></jsp:attribute>
jsp:body	Used within standard or custom tags to supply the tag body. This tag is used to define the XML dynamically i.e., the elements can generate during request time than compilation time. It actually defines the XML, which is generated dynamically element body. Syntax: <jsp:body></jsp:body>
jsp:param	Adds parameters to the request object. Syntax of <jsp:param>: <jsp: param name="param_name" value="value_of_parameter" />
jsp:text	The <jsp:text> is utilized to layout text in JSP pages. Its body does not contain any other elements, and it contains only text and EL expressions. Syntax: <jsp:text>Welcome to BAOU</jsp:text>

Check your progress 3

1. In JSP Action tags which tags are used for bean development?
 a. jsp:useBean
 b. jsp:setPoperty
 c. jsp:getProperty
 d. All of these
2. Which of the following is correct for directive in JSP?
 a. <%@directive%>
 b. <%directive%>
 c. <%!directive%>
 d. None of these
3. _____ action tag helps embeds another components such as applet.
 a. jsp:plugin
 b. jsp:config
 c. jsp:setProperty
 d. jsp:fallback

2.5 Let Us Sum Up

In this unit we have learnt that using JSP implicit objects we can easily access the data and parameters without writing unnecessary code. We also discusses JSP directives, which allows us to specify different attributes whenever required at page level, include the file and use the tag library. We also discussed Standard Action tags which are applied inside JSP pages. Such tags are used to remove or eliminate scriptlet code from JSP page as scriplet code are technically not recommended these days. Various tag libraries are used to make JSP code more compact and manageable.

2.6 Answer for Check Your Progress

Progress 1: 1. b 2. a 3. b

Progress 2: d

Progress 3: 1. d 2. a 3. a

2.7 Glossary

JavaBean: JavaBeans are classes which encapsulate several objects into a single object. It helps in accessing various objects from multiple places. It contains several elements like Constructors, Getter / Setter Methods and more.

Thread: Threads in Java are pre-defined classes, available in the java.lang package when you write your java programs. Generally, every program has one thread which is provided from the java.lang package.

XML: The Extensible Markup Language (XML) is a simple text-based format for storing and transporting structured information like documents, data, configuration, transactions, invoices and much more.

2.8 Assignment

1. Explain the various implicit objects of JSP.
2. Explain different JSP Directives.
3. Discuss various standard actions supported by JSP.

2.9 Activities

Understand the difference between include directive and jsp:include action and also implement it.

2.10 Case Study

Study and analyse MVC architecture to build Hostel Management System.

2.11 Further Readings

- https://www.tutorialspoint.com/jsp/jsp_overview.htm
- https://www.guru99.com/jsp-tutorial.html
- https://dotnettutorials.net/lesson/jsp-implicit-objects/
- https://beginnersbook.com/2013/05/jsp-tutorial-introduction/

UNIT 3: JSP EXPRESSION LANGUAGE (EL)

Unit Structure

3.0 Learning Objectives

After learning this Unit, you will be:

- Able to write Expression Language
- Able to use different Implicit Objects
- Able to use various operators through EL
- Able to call functions through EL

3.1 Introduction

Expression Language (EL) was introduced in JSP 2.0 version. Expression Language is mainly used to eliminate java code from the JSP. It is the easiest way of invoking java code. This expressions increase JSP code readability and reusability. These expressions are always within curly braces and prefixed with the dollar sign. Each EL expression is evaluated to a single value that is then expressed as text in the output of the JSP or passed as a value to a JSP action. The EL is intended to replace the need for Java (scriptlets and expressions) in JSP pages, resulting in pure JSP templates. The syntax of EL was inspired by the JavaScript (ECMAScript) syntax.

3.2 Syntax

EL expressions are always within curly braces and prefixed with the dollar sign.

- ${expr}
- ${ expr. expr }
- ${ expr [expr] }

Where, expr can be any one of the following:

- Java expression which results into any value.
- Java variable containing any value.

In EL there are two access operators.

- . (dot) operator

  It can be used in accessing values of Map having Map key or Bean having Bean property. For example, ${Map.MapKey}.
- [ ] (bracket) operator

It can be used in accessing values of Map having Map key or Bean having Bean property or List and Array having index. For example, ${Map[MapKey]},${Array[5]}

Example:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP Expression Language Usage</title>
</head>
<body bgcolor="Cyan">
            ${500}<br>
            ${24.23}<br>
            ${"Welcome to BAOU"}<br>
            ${false}<br>
            ${8+8}<br>
            ${25/5}<br>
            ${25>35}
</body>
</html>
```

EL expressions are always evaluated from left to right. The following example demonstrates how to write the expression to perform a + b.

Example:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP Expression Language Usage</title>
</head>
<body bgcolor="Cyan">
   <h1>Expression Evaluation</h1>
   <%
      request.setAttribute("a", 15);
      request.setAttribute("b", 25);
   %>
   <p>Expression Statement output</p> </br>
   <p>The Summation of A + B is: ${a+b}</p>
</body>
</html>
```

Expressions can be concatenated and that will be evaluated from left to right. The following example demonstrates how to concatenate two expressions.

Example:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP Expression Language Usage</title>
</head>
<body bgcolor="Cyan">
   <h1>Expression Concatenation</h1>
   <%
      request.setAttribute("p", 15);
      request.setAttribute("q", 25);
      request.setAttribute("r", 35);
      request.setAttribute("s", 45);
```

```
  %>
  <p>Expression Concatenation output: ${p+q}${r+s}</p>
</body>
</html>
```

In the above example you can see that the attribute values are added together within each expression and then concatenated together as a string.

Check your progress 1

1. What is the main purpose of using EL?

a. To remove XML from JSP pages

b. To remove standard actions from JSP pages

c. To remove Java syntax from JSP pages

d. To remove complexity from JSP pages

2. You can use EL alongside Java scripting elements?

a. true

b. false

3. EL expressions are always evaluated from right to left?

a. true

b. false

3.3 Different EL Implicit Objects

To allow the EL to interact with the JSP page above and beyond scoped variables that are defined by the web programmer, a number of useful implicit scoped variables are pre-defined that can be used in any EL expression on a JSP page. The JSP implicit objects can be used in scripting elements to reduce Java code inside the scripting elements. The power of EL is just because of these implicit objects only. JSP Expression Language provides following list of implicit objects to eliminate java code from JSP pages:

Implicit Object	**Description**
Cookie Information	
cookie	A map containing all javax.servlet.http.Cookie objects using the names as the keys to each object.
Header Information	

header	A map containing all request headers using header names as the keys. First header value returned, for multiple values use headerValues.
headerValues	A map containing all request headers using header names as the keys. Values are held as an Array holding all values for relevant key.
Page Context Information	
pageContext	A JavaBean containing all JSP Implicit Objects.
Parameter Information	
initParam	Map containing all context initialisation parameters using parameter names as the keys.
param	Map containing all parameter value using parameter names as the keys. First parameter value returned, for multiple values use paramValues.
paramValues	Map containing all parameter value using parameter names as the keys. Values are held as an Array holding all values for relevant key.
Scoped Attributes	
applicationScope	Map containing all context attributes within the ServletContext object using attribute names as the keys.
sessionScope	Map containing all session attributes within the HttpSession object using attributes names as the keys.
requestScope	Map containing all request attributes within the HttpServletRequest object using attribute names as the keys.
pageScope	Map containing all page attributes within page scope using attribute names as the keys.

Example of param and paramValues:

In this example, we have created two files index.jsp and user.jsp. The index.jsp file gets input from the user and sends the request to the user.jsp which in turn displays the value provided by users using EL:

index.jsp

```
<!DOCTYPE html>

<html>
```

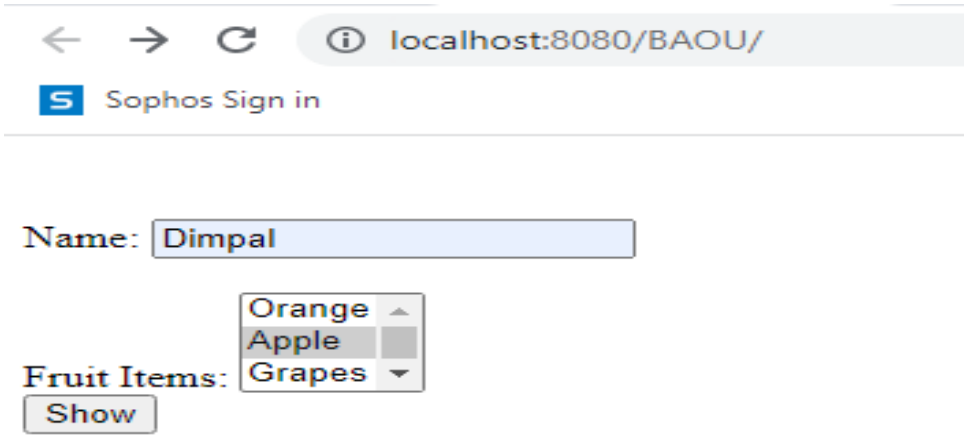
```
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
 <form method="get" action="user.jsp">
    <br>
    <br> Name: <input type="text" name="uname" /><br>
    <br> Fruit Items: <select size="4" multiple="true" name="friut">
     <option value="Orange">Orange</option>
     <option value="Apple">Apple</option>
     <option value="Grapes">Grapes</option>
     <option value="Mango">Mango</option>
    </select>
          </br>
          <input type="submit" value="Show" />
   </form>
 </font></b>
</body>
</html>
```

User.jsp

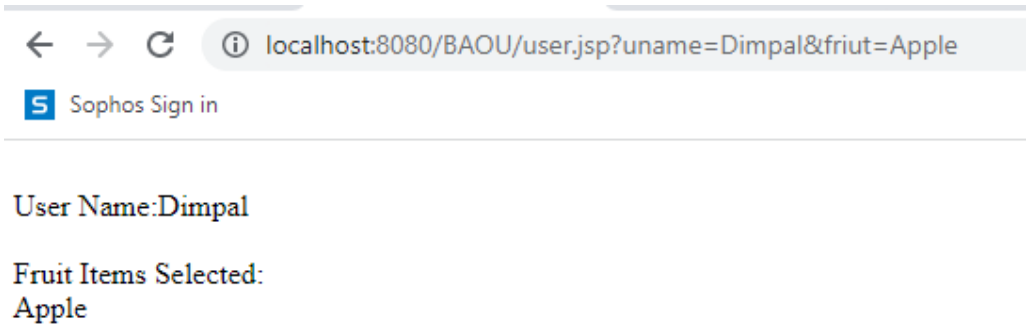
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
  <br> User Name: ${param.uname} <br>
   <br> Fruit Items Selected:<br> ${paramValues.friut[0]}
<br> ${paramValues.friut[1]}
```

```
<br> ${paramValues.friut[2]}
<br> </font></b>
</body>
</html>
```

Output:



After providing name and fruit when a user click on Show button following output will be displayed.



Examples of SessionScope:

In this example, we are printing the data stored in session scope by using Expression Language. We are using SessionScope object for printing the data.

index.jsp

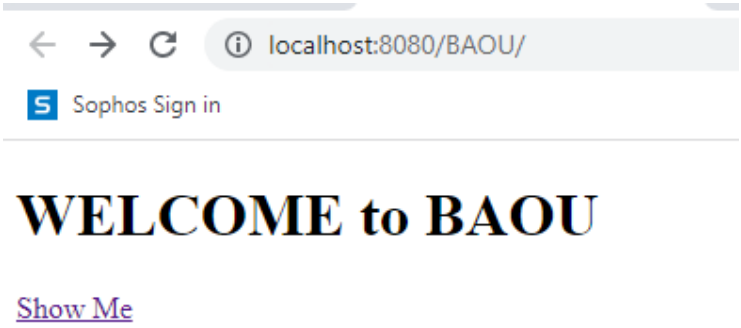
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<%@ page import="java.io.*,java.util.*"%>
<!DOCTYPE html>
```

```
<html>
<head>
<meta charset="ISO-8859-1">
<title>SessionScope Example</title>
</head>
<body>
 <h1>WELCOME to BAOU</h1>
 <%
 session.setAttribute("user", "Ved");
 %>
 <a href="user.jsp">Show Me</a>
</body>
</html>
```

user.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>SessionScope Example</title>
</head>
<body>
<h1>User Name is: ${ sessionScope.user }</h1>
</body>
</html>
```

Output:



Once user clicks on Show Me following page will be displayed.

Examples of Cookie:
Here in this example, we are printing the data stored in cookies by using cookie object.

index.jsp

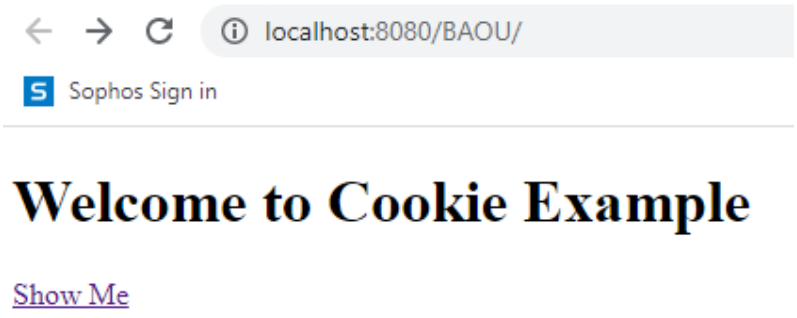
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<%@ page import="java.io.*,java.util.*"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Cookie Example</title>
</head>
<body>
<h1>Welcome to Cookie Example</h1>
<%
Cookie user=new Cookie("name","Ved");
response.addCookie(user);
%>
<a href="user.jsp">Show Me</a>
```

```
</body>
</html>
```

user.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Cookie Example</title>
</head>
<body>
<h1> Welcome to Cookie Example Mr. / Mrs.</h1>
<h2>${cookie.name.value}</hr>
</body>
</html>
```

Output:



Once user clicks on Show Me following page will be displayed.

Example of ApplicationScope

In this example, we are setting the attributes using application object and displaying those attributes using application scope.

index.jsp

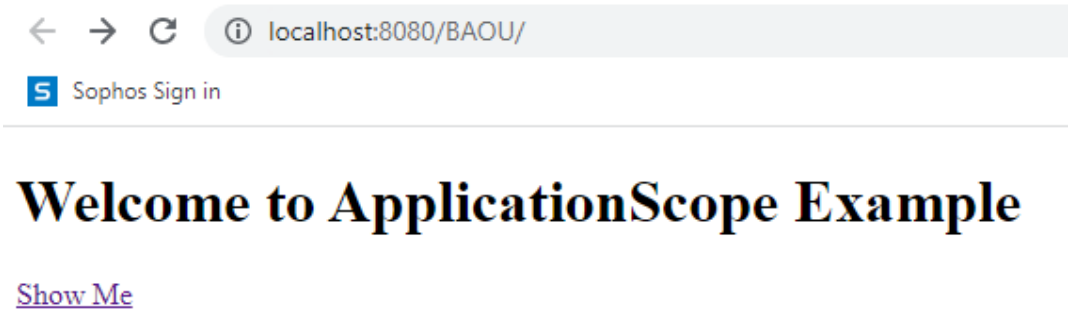
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<%@ page import="java.io.*,java.util.*"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>ApplicationScope Example</title>
</head>
<body>
<h1>Welcome to ApplicationScope Example</h1>

<%
 application.setAttribute("developer", "Ved");
 application.setAttribute("WebSite", "https://baou.edu.in/");
 %>

<a href="user.jsp">Show Me</a>
</body>
</html>
```

user.jsp

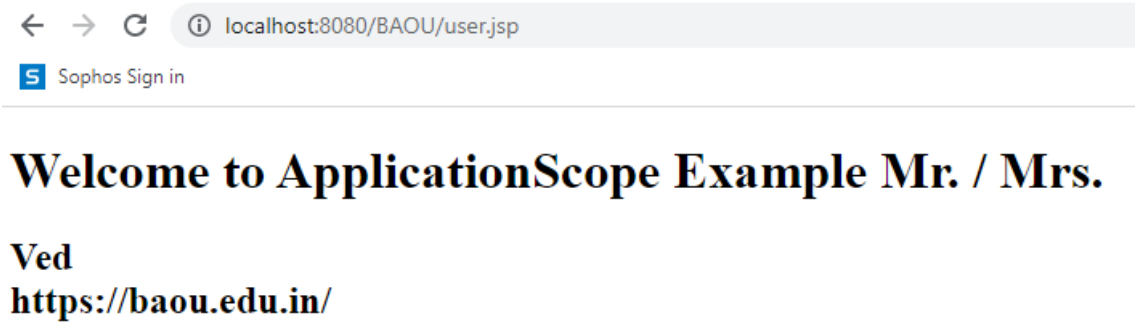
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>ApplicationScope Example</title>
</head>
<body>
```

```
<h1> Welcome to ApplicationScope Example Mr. / Mrs.</h1>
<h2>${applicationScope.developer}<br> ${applicationScope.WebSite}</hr>
</body>
</html>
```

Output:



Once user clicks on Show Me following page will be displayed.



Examples of Header

The following Example shows use of header implicit object.

index.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<%@ page import="java.io.*,java.util.*"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Using header Implicit Object Example</title>
</head>
```

```
<body>
<h1>Welcome to Header Implicit Example</h1>
${header.accept}<br>
${header["accept-encoding"]}<br>
${header["accept-language"]}<br>
${header["host"]}
</body>
</html>
```

Output:

Examples of headerValues

The following Example shows use of headerValues implicit object.

index.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<%@ page import="java.io.*,java.util.*"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Using headerValues  Implicit Object Example</title>
</head>
<body>
<h1>Welcome to headerValues  Implicit Example</h1>

${headerValues.accept[0]}<br>
```

```
${headerValues.accept[1]}<br>
${headerValues["accept-charset"][2]}<br>
${headerValues["content-type"][1]}
</body>
</html>
```

Output:

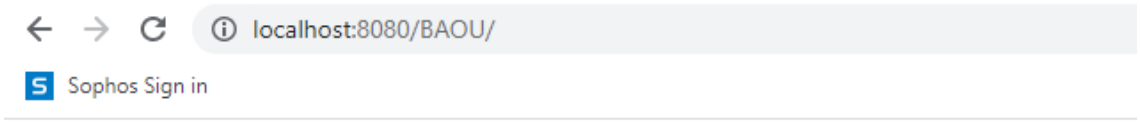
Examples of pageContext

The following Example shows use of **pageContext** implicit object.

index.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<%@ page import="java.io.*,java.util.*"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Using pageContext  Implicit Object Example</title>
</head>
<body>
<h1>Welcome to pageContext  Implicit Example</h1>
${pageContext.request.servletPath}<br>
${pageContext.request.method}<br>
${pageContext.session.id}
</body>
</html>
```

Output:



Welcome to pageContext Implicit Example

/index.jsp
GET
D673CF41A6284B7255BF7B7FF50E31B0

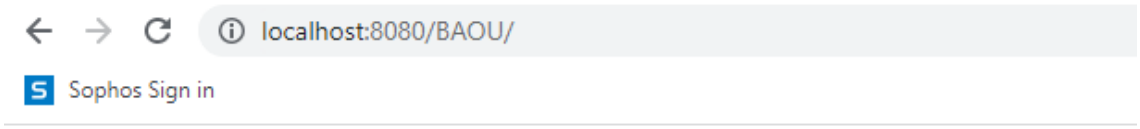
Examples of requestScope

The following example shows use of requestScope implicit object.

index.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<%@ page import="java.io.*,java.util.*"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Using requestScope Implicit Object Example</title>
</head>
<body>
<h1>Welcome to requestScope Implicit Example</h1>
<%
    pageContext.setAttribute("name", "Ved", PageContext.REQUEST_SCOPE);
%>
${requestScope.name}<br>
${requestScope["name"]}
</body>
</html>
```

Output:



Welcome to requestScope Implicit Example

Ved
Ved

Examples of pageScope

The following example shows use of pageScope implicit object.

index.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<%@ page import="java.io.*,java.util.*"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Using requestScope Implicit Object Example</title>
</head>
<body>
<title>Using pageScope Implicit Object Example</title>
</head>
<body>
<h1>Welcome to pageScope Implicit Example</h1>
<%
   pageContext.setAttribute("name", "Dimpy", PageContext.PAGE_SCOPE);
%>
${pageScope.name}<br>
${pageScope["name"]}
</body>
</html>
```

Output:

Check your progress 2

1. Which EL implicit object is not of type java.util.Map?

a. cookie

b. header

c. initParam

d. pageContext

2. Using array like syntax such as ${header.accept[1]}, we can access secondary header values using the header EL implicit object?

a. true

b. false

3. The initParam EL implicit object allows access to servlet initialization parameters?

a. true

b. false

4. Which of the following EL implicit object contains variables with the shortest scope?

a. applicationScope

b. pageScope

c. sessionScope

d. requestScope

3.4 EL Operators

There are different operators that can be used within EL expressions. Symbols used for mathematical and logical evaluation that are recognized by the compiler are generally known as operators in Java. With EL we also have a comprehensive, although reduced, list of

operators. Here, we will look at the arithmetic, relational, logical, conditional and empty EL operators that we can use within our JSP pages.

Arithmetic Operators

We are familiar with most of the arithmetic operators shown in the table below.

Operator	Meaning	Example
+	Addition	${a + b}
-	Subtraction	${a - b}
/ div	Division	${a / b} ${a div b}
*	Multiplication	${a * b}
% mod	Modulus	${a % b} ${a mod b}

Example:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP Expression Language Usage</title>
</head>
<body bgcolor="Cyan">
   <h1>Arithmetic Operator</h1>
  <%
     request.setAttribute("a", 15);
     request.setAttribute("b", 25);
   %>
   <p>Expression Statement output</p> </br>
   <p>The Summation of A + B is: ${a+b}</p> </br>
```

```
        <p>The Subtraction of A - B is: ${a-b}</p> </br>
        <p>The Division of A / B is: ${a/b}</p> </br>
        <p>The Division of A div B is: ${a div b}</p> </br>
        <p>The Multiplication of A * B is: ${a * b}</p> </br>
        <p>The Modulus of A % B is: ${a % b}</p> </br>
        <p>The Modulus of A mod B is: ${a mod b}</p> </br>
</body>
</html>
```

Relational Operators

Relational operator takes two operands, compare their values and returns a Boolean value (either true or false). These operators are typically used in conditional expressions to check whether a condition is true or not. The table below contains various relational operators.

Operator	Meaning	Example	Result	Remarks
== eq	Equal to	${a == b} ${a eq b}	true	All types can be compared for equality
!= ne	Not Equal to	${a != b} ${a ne b}	false	All types can be compared for inequality
< lt	Less than	${a < b} ${a lt b}	false	It can be used with all numeric types and the char type.
<= le	Less than or equal to	${a <= b} ${a le b}	true	It can be used with all numeric types and the char type.
> gt	Greater than	${a > b} ${a gt b}	false	It can be used with all numeric types and the char type.
>= ge	Greater than or equal to	${a >= b} ${a ge b}	true	It can be used with all numeric types and the char type.

Example:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP Expression Language Usage</title>
</head>
<body bgcolor="Cyan">
   <h1>Relational Operator</h1>
  <%
     request.setAttribute("a", 15);
     request.setAttribute("b", 25);
   %>
   <p>Expression Statement output</p> </br>
   <p>The Equality operator: ${a == b}</p> </br>
         <p>The Equality operator: ${a eq b}</p> </br>

         <p>The Un-Equality operator: ${a ne b}</p> </br>
         <p>The Un-Equality operator: ${a != b}</p> </br>

         <p>The Less Than operator: ${a < b}</p> </br>
         <p>The Less Than operator: ${a lt b}</p> </br>

         <p>The Greater Than operator: ${a > b}</p> </br>
         <p>The Greater Than operator: ${a gt b}</p> </br>

         <p>The less than or equal operator: ${a le b}</p> </br>
         <p>The less than or equal operator: ${a <= b}</p> </br>

         <p>The greater than or equal operator: ${a ge b}</p> </br>
         <p>The greater than or equal operator: ${a >= b}</p> </br>

</body>
</html>
```

Logical Operators

Logical Operands must be the Boolean type and the result of a logical operation is the Boolean type and are used with control statements. The following table shows all logical operators.

Operator	Meaning	Result	Remarks
&& and	Logical AND	false / true	If the first operand returns false, the second operand will not be checked (short-circuited) and false is returned.
|| or	Logical OR	False / true	If the first operand returns true, the second operand will not be checked (short-circuited) and true is returned.
! not	Logical NOT	True / false	Will check if operand is not true.

Example:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP Expression Language Usage</title>
</head>
<body bgcolor="Cyan">
   <%
     request.setAttribute("a", 15);
     request.setAttribute("b", 25);
     Boolean p = false;
     Boolean q = true;
   %>
   <p>Expression Statement output</p> </br>
   <p>Logical AND Expression Result: ${(a<b)&&(6<12)} </p></br>
         <p>Logical AND Expression Result: ${(a<b)and(6>12)} </p></br>
```

```
        <p>Logical OR Expression Result: ${(a<b)||(6<12)} </p></br>
        <p>Logical OR Expression Result: ${(a<b)or(6>12)} </p></br>

        <p>Logical NOT Expression Result: ${!p} </p></br>
        <p>Logical NOT Expression Result: ${not p} </p></br>
</body>
</html>
```

The Conditional Operator

The Conditional operator is a ternary operator and can be used to replace an if....else construct. It takes three operands as shown below:

Block	**Meaning**
if....else	

if (condition) {
  variable = expression1;
} else {
  variable = expression2;
}
	Assign result of expression1 to variable
	if condition evaluates to true,
	
	otherwise assign result
	of expression2 to variable.

Conditional Operator	
${statement? expression1 : expression2	If statement evaluates to true evaluate expression1, otherwise evaluate expression2,

The following example shows how to use the Conditional operator. It results in Small Number being output:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

   pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

```
<title>JSP Expression Language Usage</title>

</head>

<body bgcolor="Cyan">

   <h1>Conditional Operator</h1>

<%

  request.setAttribute("a", 25);

  request.setAttribute("b", 35);

%>

<p>Conditional Expression Result: ${(requestScope.a lt requestScope.b)? "Small Number" :
"Large Number"} </p></br>

<p>Conditional Expression Result: ${(9 + 7) > 11 ? (12 + 13) : (14 + 15) } </p></br>

</body>

</html>
```

The empty Operator

The empty operator can be used to check the emptiness of an object and has the following syntax:

- ${empty object}

The empty operator will return true when any of the following criteria is satisfied otherwise it will return false:

- If the object being tested is null.
- If the object being tested is an empty string ("").
- If the object being tested is an empty array.
- If the object being tested is an empty Map or Collection.

Example:

```
<%@ page language="java" import="java.util.List,java.util.Arrays" contentType="text/html;
charset=ISO-8859-1"   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
```

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP Expression Language Usage</title>
</head>
<body bgcolor="Cyan">
   <h1> Empty Operator</h1>
  <%
         List  country = Arrays.asList("USA", "AUSTRALIA", "INDIA",
"JAPAN","ISREAL");
         pageContext.setAttribute("Country", country);
      %>
   <p>Empty Expression Statement </p> </br>
   <p>Empty Expression Result: ${empty country} </p></br>
</body>
</html>
```

Operator Precedence

Following is the precedence of operators, having priority descends from top to bottom and from left to right in a row:

- [] .
- ()
- - (unary) not ! empty
- / div % mod
- + - (binary)
- < > <= >= lt gt le ge
- == != eq ne
- && and
- || or
- ? :

Examples:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP Expression Language Usage</title>
</head>
<body bgcolor="Cyan">
   <h1> Operator Precedence</h1>
   <p>Operator Precedence one: ${5 * 6 + 2 * 8} </p></br>
         <p>Operator Precedence two: ${10 + 12 / 4 - 2} </p></br>
         <p>Operator Precedence three: ${6 > 4 && 5 > 7} </p></br>
         <p>Operator Precedence four: ${2 + 3 > 4 ? 5 * 6 : 7 - 8} </p></br>
</body>
</html>
```

Check your progress 3

1. What will be the output from the following expression?

```
<%
  request.setAttribute("a", 15);
  request.setAttribute("b", 6);
%>
${a%b}
```

a. 9

b. -3

c. 3

d. -9

2. When using the Short-circuit AND (and or &&) if the first operand returns false, the second operand will not be checked?

a. true

b. false

3. When using the Short-circuit OR (or or ||) if the first operand returns true, the second operand will not be checked?

a. true

b. false

4. What will be the output from the following expression?

${empty ""}

a. an exception will be thrown

b. null

c. true

d. false

5. Which operator has the highest precedence?

a. not

b. instanceof

c. mod

d. empty

3.5 Let Us Sum Up

In this unit we have learnt that Expression Language is used to access the data. IT helps developer to easily access the application data stored in objects like Java Beans, request, session and application etc. We also discussed implicit objects through which developers can get parameter values and attributes from different scopes. Apart from this we also discussed different operator's usage and how it helps developer to write a JSP application efficiently and effectively without much of the java code hassle.

3.6 Answer for Check Your Progress

Progress 1: 1. c 2. a 3. b

Progress 2: 1. d 2. b 3. b 4. b

Progress 3: 1. c 2. a 3. a 4. c 5. a

3.7 Glossary

ECMAScript: ECMAScript is also known as JavaScript. It is a programming language adopted by the European Computer Manufacturer's Association as a standard for performing computations in Web applications. Languages like JavaScript, Dart-lang and C# are standardized by ECMA.

Scoped Variable: The scope defines the order in which variable names are resolved, the lifetime of the variable and its purview.

3.8 Assignment

1. Define EL and discuss its benefits.
2. Discuss various implicit objects of Expression language.
3. Write a short note on Operator in Expression Language.

3.9 Activities

Understand and implement function in Expression Language.

3.10 Further Readings

- https://www.tutorialspoint.com/jsp/jsp_overview.htm
- https://www.javatpoint.com/EL-expression-in-jsp
- https://server2client.com/el/elimpobjs.html
- https://www.h2kinfosys.com/blog/expression-language/

UNIT 4: JSTL

Unit Structure

4.0 Learning Objectives

After learning this Unit, you will be:

- Able to define various JSTL tags and its usage
- Write a JSP file to replace a Java scripting completely from our JSP pages by using JavaBeans and EL in conjunction with JSTL
- Use JSTL functions

4.1 Introduction

JSP Standard Tag Library (JSTL) is a standard tag library of readymade tags. The JSTL library contains several tags that will remove JSP scriptlet code from a JSP page by providing already implemented common functionalities. It is a collection of useful JSP tags that encapsulates the basic functionality common to many JSP applications. It supports common, structural tasks like iteration and conditions, tags for manipulating XML documents, internationalization tags and SQL tags. JSTL also supports a structure for integrating the existing custom tags with the JSTL tags.

JSTL delivers various advantages as mentioned below**:**

1. It makes easy for a developer to understand and write the code.
2. As JSP scriptlets confuse programmers, the usage of JSTL makes the code neat and clean.
3. JSTL Expression language handles JavaBean property very efficiently.
4. Based on XML, which is similar to HTML, JSTL is very easy for the programmers to understand.

4.2 JSP Custom Tag Library

As being a part of the Java EE API, JSTL is generally included in most servlet containers. To use JSTL in the JSP pages, user need to download the JSTL jars for the servlet container. Mostly, these jars are available in the example folder of the server downloaded. If these jars (libraries) are not available then separately downloads it and transfer it to web application project WEB-INF/lib directory.

JSTL jars

JSTL jars are container specific, means in Apache Tomcat, you need to include jstl.jar and standard.jar jar files in your project build path. If they are not present in the container lib directory, you should include them into your application. The JAR files must be copied into the WEB-INF/lib directory of every web application that requires JSTL or into the library directory of the container you are using.

The JSTL is subdivided into 5 groups including Core, XML, I18N, Database and Functions libraries where actions are grouped together by category. The following table demonstrates the tag libraries and the functional areas along with the URI location and the preferred prefix:

Tag Library	**Functional Areas**	**URI**	**Prefix**
Core	It provide support for Variable Support, iteration, conditional logic, catch an exception, URL management, forward or redirect response, Miscellaneous.	http://java.sun.com/jsp/jstl/core	c
XML	These tags are used to work with XML documents such as parsing XML, transforming XML data and XPath expressions evaluation, Core Flow control and Transformation.	http://java.sun.com/jsp/jstl/xml	x
Ii8n / Formatting	Formatting library provides tags to format text, date, number for Internationalised web sites.	http://java.sun.com/jsp/jstl/fmt	Fmt
Database	Database library provides support for Relational Database Connection like Oracle, MySQL etc. Allows interaction to perform operations like insert, delete, update, select etc on SQL databases.	http://java.sun.com/jsp/jstl/sql	Sql
Functions	JSTL tags provide a number of functions that we can use to perform a common operation, mostly for String manipulation such as String Concatenation, String Spliting etc.	http://java.sun.com/jsp/jstl/functions	fn

All the JSTL standard tags URI starts with https://java.sun.com/jsp/jstl/ and you can use any prefix you want. But it is advisable to use the above defined prefix to avoid any confusion because they are the most widely used prefixes.

Syntax to include JSTL functions in JSP page is:

<%@taglib uri="uri" prefix="prefix" %>

Check your progress 1

1. To use JSTL functionality, we need a(n) _____ container.

a. Servlet

b. HTML

c. JSP

d. JavaScript

4.3 JSTL Core Tags

JSTL Core Tags are listed in the following table.

JSTL Core Tag	Description
<c:out>	To write something in JSP page, we can use EL also with this tag
<c:import>	Same as jsp:include or include directive
<c:redirect>	redirect request to another resource
<c:set>	To set the variable value in given scope.
<c:remove>	To remove the variable from given scope

JSTL Core Tag	Description
<c:catch>	To catch the exception and wrap it into an object.
<c:if>	Simple conditional logic, used with EL and we can use it to process the exception from <c:catch>
<c:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <c:when> and <c:otherwise>
<c:when>	Subtag of <c:choose> that includes its body if its condition evalutes to 'true'.
<c:otherwise>	Subtag of <c:choose> that includes its body if its condition evalutes to 'false'.
<c:forEach>	for iteration over a collection
<c:forTokens>	for iteration over tokens separated by a delimiter.
<c:param>	used with <c:import> to pass parameters
<c:url>	to create a URL with optional query string parameters

Syntax:

<%@ taglib prefix=”c” uri=”http://java.sun.com/jsp/jstl/core”%>

Example: following example demonstrates the use of important core tags.

```
<%@ taglib uri=”http://java.sun.com/jsp/jstl/core” prefix=”c” %>
<html>
 <head>
  <title>Welcome to BAOU</title>
 </head>
 <body>
       <h1> Welcome to BAOU </h1>
  <b><c:out value = "${'<p>c:out is used to display the result of an expression in
  the web browser.</p>'}" default="Not Available" escapeXml="true"/></b></br>
  <c:import var="data" url="header.jsp" charEncoding="UTF-8" />
  <c:out value = "${data}"/>
 <c:set var = "user" scope = "session" value="${2023}" /><br>
  The value defined in the session scope is: <c:out value = "${user}"/></br>
 <c:remove var="user" scope="session"/><br>
  The value after removing is: <c:out value = "${user}"/></br>
  <c:catch var ="exceCaught">
       <% int v = 8/0; %>
 </c:catch>
<c:if test = "${exceCaught != null}">
       <p>Exception is: ${exceCaught}</p></br>
       <p>Exception raised because : ${exceCaught.message}</p></br>
</c:if>
<c:set var="number" value="98"/>
       <c:choose>
            <c:when test="${number > 24}">
                 Number value is greater than 24.</br>
            </c:when>
            <c:otherwise>
                 Number value is less than or equal to 24.</br>
            </c:otherwise>
```

```
        </c:choose>
     <p> Page will redirected because number is greater than 1 </p></br>
        <c:if test="${number > 1}">
                <c:redirect url="https://baou.edu.in/"/>
        </c:if>
 </body>
</html>
```

Check your progress 2

1. JSTL core tag provides support for following except

a. Conditionals

b. Manipulating text documents.

c. Url

d. Iteration

2. How many Core tag library actions are there?

a. 14

b. 12

c. 9

d. 17

3. We can use the target and property attributes of the </c:set> action to assign a new property value to any object?

a. True

b. False

4. What type of error object is thrown from the <c:catch> tag?

a. java.lang.Error

b. java.lang.Exception

c. java.lang.Throwable

d. All of the above

4.4 JSTL Function Tag

The main objective of the functions tag library is to perform all the String manipulation operations which are defined in the String class.

JSTL Function	Details
fn:contains()	It is used to test if an input string contains the specified substring in a program
fn:containsIgnoreCase()	It is used to test if an input string contains the specified substring as a case insensitive way
fn:endsWith()	It checks whether the specified string is a suffix of a given string or not
fn:length()	It returns the number of characters in a string
fn:trim()	It removes spaces from beginning and end of a string
fn:indexOf()	It returns an index within a string of first occurrence of a given substring
fn:escapeXml()	It escapes the characters that would be interpreted as XML markup
fn:startsWith()	It checks whether the specified string is a prefix of a given string or not
fn:split()	It splits the string into an array of substrings
fn:toLowerCase()	It converts all the characters of a string to lower case
fn:toUpperCase()	It converts all the characters of a string to upper case
fn:substring()	It returns the subset of a string according to the given start and end position
fn:substringAfter() fn:substringBefore()	It returns the subset of string after a specific substring It returns the subset of string before a specific substring
fn:replace()	It replaces all the occurrence of a string with another string sequence
fn:join()	The fn:join() function joins the all array elements with a specified separator.

Syntax:

<% taglib uri=http://java.sun.com/jsp/jstl/functions prefix=”fn” %>

Example: following example demonstrates the use of important function tags.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>
```

```
<html>
<head>
<title>JSTL Function Tags</title>
</head>
<body>
<h1> Welcome to BAOU </h1>
<c:set var="data" value="Welcome to Sardar Patel University"/>
        <b>Given String: </b></br>
        <c:out value="${data}" /><br/></br>
        <b>String after replacing Welcome to Most Welcome:</b></br>
        <c:out value="${fn:replace(data, 'Welcome', 'Most Welcome')}"/> </br>
        <b>Substring of the given string:</b><br/>
                <c:out value="${fn:substring(data, 11, '23')}" /> </br>
                <b>Length of the given string: </b></br>
                <c:out value="${fn:length(data)}" /> </br>

                <c:set var="testStr" value="You are in the State of Gujarat "/>
                <b>String before splitting:</b> </br>
                <c:out value="${testStr}" />
                <c:set var="StringAfterSplitting"
                     value="${fn:split(testStr, ' ')}"/></br>
                <c:set var="StringAfterJoining"
            value="${fn:join(StringAfterSplitting, '-')}"/></br>
                <b>Final String after split and join operation:</b></br>
                <c:out value="${StringAfterJoining}" /> </br>
                <b>Index of State in the String is: </b>
                <c:out value="${fn:indexOf(testStr, 'State')}" /></br>

</body>
</html>
```

Check your progress 3

JSTL function tag provides support for manipulating

a. Integer

b. Float

c. String

d. Boolean

4.5 JSTL Formatting Tag

The formatting tag provides support for message formatting, number and date formatting etc.

Syntax:

<%@ taglib uri=”http://java.sun.com/jsp/jstl/fmt” prefix=”fmt” %>

Following table contains various formatting tags:

Formatting Tags	Details
fmt:parseNumber	It is used to Parses the string representation of a currency, percentage or number.
fmt:timeZone	It specifies a parsing action nested in its body or the time zone for any time formatting.
fmt:formatNumber	It is used to format the numerical value with specific format or precision.
fmt:setTimeZone	It stores the time zone inside a time zone configuration variable.
fmt:bundle	It is used for creating the ResourceBundle objects which will be used by their tag body.
fmt:message	It display an internationalized message.
fmt:setBundle	It loads the resource bundle and stores it in a bundle configuration variable or the named scoped variable.
fmt:formatDate	It formats the time and/or date using the supplied pattern and styles.
fmt:parseDate	It parses the string representation of a time and date.

Example: following example demonstrates the use of important formatting tags.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<html>
<head>
```

```
<title>JSTL Formatting Tags</title>
</head>
<body>
<h1> Welcome to BAOU </h1>
<c:set var="TODAY" value="<%=new java.util.Date()%>"/>
        <b>Present date after setting type attribute to date: <b></br>
        <fmt:formatDate type="date" value="${TODAY}" /> </br>
        <b>Present date after setting type attribute to time: <b></br>
        <fmt:formatDate type="time" value="${TODAY}" /></br>
        <b>Present date after setting type attribute to both: <b></br>
        <fmt:formatDate type="both" value="${TODAY}" /> </br>
        <b>Present date after setting pattern attribute: <b></br>
        <fmt:formatDate pattern="yyyy-MM-dd" value="${TODAY}"/> </br>

        <c:set var="TODAY" value="2023-03-01" />
              <fmt:parseDate value="${TODAY}"
             var="parsedDate" pattern="yyyy-MM-dd" />
              <b>Current date after parsing: </b></br>
              <c:out value="${parsedDate}"/>      </br>

              <c:set var="number" value="2568.4589"/>
  <b>Number after parsing by setting type attribute: <b><br>
  <fmt:parseNumber  var="num" value="${number}" type="number"/>
  <c:out value="${num}"/></br>
  <b>Number after parsing by setting integerOnly attribute to true: </b></br>
  <fmt:parseNumber  var="num" value="${number}" integerOnly="true"/>
  <c:out value="${num}"/></br>

      <c:set var="TODAY" value="<%=new java.util.Date()%>"/>
              <fmt:formatDate value="${TODAY}" type="both"/> <br/>

              <b>Set Time Zone to HST </b></br>
              <fmt:setTimeZone value="HST" />
```

```
                <fmt:formatDate value="${TODAY}" type="both"/>          </br>


</body>
</html>
```

Check your progress 4

JSTL formatting tag provides support for numbers, dates and i18n.

a. True

b. False

4.6 JSTL XML Tag

The XML tags are used for providing a JSP centred way of manipulating and creating XML documents. The XML tag library is based on XPath, which provides a concise notation for specifying and selecting parts of an XML document. XPath can be thought of as an expression language.

Syntax:

<%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x" %>

Following table contains various XML tags:

XML Tags	Details
x:out	It is similar to <%= ... > tag, but for XPath expressions.
x:parse	It is used for parse the XML data specified either in the tag body or an attribute.
x:choose	It is a conditional tag that establish a context for mutually exclusive conditional operations.
x:set	It is used to sets a variable to the value of an XPath expression.
x:when	It is a subtag of that will include its body if the condition evaluated be 'true'.
x:if	It is used for evaluating the test XPath expression and if it is true, it will processes its body content.
x:otherwise	It is subtag of that follows tags and runs only if all the prior conditions evaluated be 'false'.

x:transform	It is used in a XML document for providing the XSL(Extensible Stylesheet Language) transformation.
x:param	It is used along with the transform tag for setting the parameter in the XSLT style sheet.

To utilize the functionality of XML and XPath related libraries, it is required to download and add following three jar files within the lib directory of your project.

- XercesImpl.jar
- xalan.jar
- xml-apis-1.4.01.jar

Example: following example demonstrates the use of important XML tags.

```
<%@ taglib  uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib  uri="http://java.sun.com/jsp/jstl/xml" prefix="x"%>
<html>
<head>
<title>JSTL  XML Tags</title>
</head>
<body>
<h1> Welcome to BAOU </h1>
<h2>JSTL XML Tag Example </h2>
<c:set var="University">
<University>
    <name>Sardar Patel University</name>
    <city>Vallabh Vidyanagar</city>
</University>
</c:set>
<x:parse xml="${University}" var="data"/>
<b>Website :: </b>
<x:out select="$data/University/name" /></br>
<b>Tutorial ::</b>
<x:out select="$data/University/city" /> </br>
```

```
            <c:set var = "xmlData">
      <books>
        <book>
          <name>Data Visualization</name>
          <author>Vinod</author>
          <price>250</price>
        </book>

        <book>
          <name>Machine Learning</name>
          <author>Ajay</author>
          <price>400</price>
        </book>
      </books>
    </c:set>

    <x:parse xml = "${xmlData}" var = "output"/>
    <x:choose>
      <x:when select = "$output//book/author = 'Vinod'">
        Book is Authored by Dr. Vinod
      </x:when>

      <x:when select = "$output//book/author = 'Ajay'">
        Book is written by Dr. Ajay
      </x:when>
      <x:otherwise>
        Author is not Known.
      </x:otherwise>
    </x:choose>
</body>
</html>
```

Check your progress 5

1. JSTL xml tag provides support for which of the following,

a. parsing XML

b. Transforming XML

c. XPath Expression Evaluation

d. All of these

2. How many XML tag library actions are there?

a. 7

b. 10

c. 8

d. 14

3. Which XML attribute allows us to specify XPath expressions?

a. allow

b. select

c. transform

d. view

4.7 JSTL SQL Tag

The SQL tags provide SQL support. This tag library allows the tag to interact with RDBMSs like Microsoft SQL Server, mySQL or Oracle.

Syntax:

<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>

Following table contains various SQL tags:

SQL Tags	Details
sql:setDataSource	It is used for creating a simple data source suitable only for prototyping.
sql:param	It is used for sets the parameter in an SQL statement to the specified value.
sql:query	It is used for executing the SQL query defined in its sql attribute or the body.
sql:update	It is used for executing the SQL update defined in its sql attribute or in the tag body.

sql:transaction	It is used to provide the nested database action with a common connection.
sql:dateParam	It is used for sets the parameter in an SQL statement to a specified java.util.Date value.

To utilize the functionality of SQL libraries, it is required to download and add following mysql jar file within the lib directory of your project to connect the mysql database.

- mysql-connector.jar

Example: following example demonstrates the use of important SQL tags.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
 <head>
   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
   <title>sql:query Example</title>
 </head>
 <body>
   <h1>Welcome to BAOU</h1>
  <h3>SQL:query Example</h3>
   <sql:setDataSource var="myDS" driver="com.mysql.jdbc.Driver"
     url="jdbc:mysql://localhost:3306/mca" user="root"  password=""/>
     <sql:query dataSource="${myDS}" var="students">
     SELECT * from student;
   </sql:query>
   <table border="1">
     <c:forEach var="row" items="${students.rows}">
       <tr>
         <td><c:out value="${row.sno}"/></td>
```

```
        <td><c:out value="${row.sname}"/></td>
        <td><c:out value="${row.smob}"/></td>
        </tr>
    </c:forEach>
  </table>
 </body>
</html>
```

Check your progress 6

1. JSTL SQL tag provides support to interact with non relational databases.

a. True

b. False

2. Which action is used for date parameterisation?

a. <sql:date>

b. <sql:param>

c. <sql:dateParam>

d. <sql:value>

3. Which Database action allows us to safeguard database integrity?

a. <sql:commit>

b. <sql:rollback>

c. <sql:update>

d. <sql:transaction>

4.8 Let Us Sum Up

In this unit we learnt that JSP Standard Tag Library(JSTL) is a standard library of readymade tags which replaces the scriptlet code from a JSP page by providing some ready to use functionalities. We discussed various important functionality relating core tags such as if, forEach, import, out etc to support some basic scripting task. Formatting library provides tags to format text, date, number for internationalized web sites. SQL library provides support for Relational Databases and tags to perform operations like insert, delete, update, select on SQL databases. XML tag provides support for XML processing like flow control, transformation features etc. Functions tag provides support for string manipulation.

4.9 Answer for Check Your Progress

Progress 1: a

Progress 2: 1. b 2. a 3. b 4. c

Progress 3: c

Progress 4: a

Progress 5: 1. d 2. b 3. b

Progress 6: 1. b 2. c 3. d

4.10 Glossary

Taglib: The taglib directive specifies that the JSP page uses a set of custom tags, identifies the location of the library.

Prefix: It is a letter or group of letters added to the start of a word to change the meaning or make a new word.

JSTL: Group of different Tag Libraries

4.11 Assignment

1. Explain various benefits of JSTL?
2. Discuss various JSTL core tags.
3. Discuss various JSTL String manipulation tags.
4. Discuss various JSTL xml tags.

4.12 Activities

Understand and implement various JSTL SQL tags.

4.13 Case Study

Study and analyse MVC architecture to build Hostel Management System.

4.14 Further Readings

- https://www.baeldung.com/jstl
- https://server2client.com/jstl/jstl.html
- https://www.guru99.com/jsp-tag-library.html
- https://www.digitalocean.com/community/tutorials/jstl-tutorial-jstl-tags-example

BAOU
Education for All

યુનિવર્સિટી ગીત

સ્વાધ્યાયઃ પરમં તપઃ
સ્વાધ્યાયઃ પરમં તપઃ
સ્વાધ્યાયઃ પરમં તપઃ

શિક્ષણ, સંસ્કૃતિ, સદ્‌ભાવ, દિવ્યબોધનું ધામ
ડૉ. બાબાસાહેબ આંબેડકર ઓપન યુનિવર્સિટી નામ;
સૌને સૌની પાંખ મળે, ને સૌને સૌનું આભ,
દશે દિશામાં સ્મિત વહે હો દશે દિશે શુભ-લાભ.

અભણ રહી અજ્ઞાનના શાને, અંધકારને પીવો ?
કહે બુદ્ધ આંબેડકર કહે, તું થા તારો દીવો;
શારદીય અજવાળા પહોંચ્યાં ગુર્જર ગામે ગામ
ધ્રુવ તારકની જેમ ઝળહળે એકલવ્યની શાન.

સરસ્વતીના મયૂર તમારે ફળિયે આવી ગહેકે
અંધકારને હડસેલીને ઉજાસના ફૂલ મહેંકે;
બંધન નહીં કો સ્થાન સમયના જવું ન ઘરથી દૂર
ઘર આવી મા હરે શારદા દૈન્ય તિમિરના પૂર.

સંસ્કારોની સુગંધ મહેંકે, મન મંદિરને ધામે
સુખની ટપાલ પહોંચે સૌને પોતાને સરનામે;
સમાજ કેરે દરિયે હાંકી શિક્ષણ કેરું વહાણ,
આવો કરીયે આપણ સૌ
ભવ્ય રાષ્ટ્ર નિર્માણ...
દિવ્ય રાષ્ટ્ર નિર્માણ...
ભવ્ય રાષ્ટ્ર નિર્માણ

DR. BABASAHEB AMBEDKAR OPEN UNIVERSITY
(Established by Government of Gujarat)
'Jyotirmay' Parisar,
Sarkhej-Gandhinagar Highway, Chharodi, Ahmedabad-382 481
Website : www.baou.edu.in