



**DR. BABASAHEB AMBEDKAR  
OPEN UNIVERSITY**

# BCA



**BCAR-502**

**Object Oriented Analysis and Design**

# **OBJECT ORIENTED ANALYSIS AND DESIGN**



**DR. BABASAHEB AMBEDKAR OPEN UNIVERSITY  
AHMEDABAD**

## **Editorial Panel**

**Author : Dr. Tulsidas Nakrani**  
**Associate Professor**  
**Sankalchand Patel University**  
**Visnagar**

**Editor : Dr. Parimalkumar P Patel**  
**I/c Principal**  
**Khyati School of Computer Application, BCA**  
**(Affiliated to Gujarat University)**

**Language Editor : Dr. Prakash Khuman**  
**Principal**  
**Lokmanya College of Commerce**  
**Gujarat University**

**ISBN 978-93-91071-07-3**

**Edition : 2022**

**Copyright © 2020 Knowledge Management and Research Organisation.**

All rights reserved. No part of this book may be reproduced, transmitted or utilized in any form or by a means, electronic or mechanical, including photocopying, recording or by any information storage or retrieval system without written permission from us.

## **Acknowledgment**

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

## **ROLE OF SELF-INSTRUCTIONAL MATERIAL IN DISTANCE LEARNING**

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

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

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

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

Teaching and learning at a distance eliminate interactive communication cues, such as pauses, intonation and gestures, associated with the face-to-face method of teaching. This is

particularly so with the exclusive use of print media. Instructional activities built into the instructional repertoire provide this missing interaction between the student and the teacher. Therefore, the use of instructional activities to affect better distance teaching is not optional, but mandatory.

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

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

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

## **PREFACE**

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

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

All the best for your studies from our team!

# OBJECT ORIENTED ANALYSIS AND DESIGN

## Contents

---

### **BLOCK 1 : FUNDAMENTALS OF SYSTEM ANALYSIS AND DESIGN**

---

#### **Unit 1 FUNDAMENTALS OF SYSTEM ANALYSIS AND DESIGN**

Introduction, Types of Systems, Transaction Processing System (TPS), Management Information System (MIS), Decision Support System (DSS), Role of System Analyst, System Development Life Cycle, CASE Tools, Interviewing

#### **Unit 2 ANALYSIS MODELING**

Introduction, Data Flow Diagram, History of the DFD, Symbols and Notations Used in DFDs, DFD Rules and Tips, DFD Levels and Layers : From Context Diagrams to Pseudo Code, Examples of DFD Use, DFD and Unified Modeling Language (UML), Logical Vs. Physical DFD, Data Dictionary, Types of Data Dictionaries, Data Dictionary Components, How to Create a Data Dictionary, Pros and Cons of Data Dictionaries, Structured English, Pseudocode, Guidelines for Choosing the Right Tools, Decision table, Components of a Decision Table, Decision Trees

#### **Unit 3 SYSTEM DESIGN**

Introduction, Design of Effective Results, Output Design Objectives, Data Concepts and Normalization, Data Warehouses, Human-Computer Interaction, Query Design, Effective Coding

---

### **BLOCK 2 : BASICS OF OBJECT ORIENTED MODELLING**

---

#### **Unit 4 INTRODUCTION TO OBJECT ORIENTATION MODELING**

Introduction, Object Oriented Modelling, Characteristics of Object Oriented Modelling, Links and Association, Generalization and Inheritance, An Object Model, Benefits of OO Modelling

**Unit 5      IMPORTANCE OF MODELING**

Introduction, Importance of Modelling, Importance of Business Domain Modelling

**Unit 6      ADVANCED MODELING CONCEPTS**

Introduction, Aggregation, Abstract Class, Multiple Inheritance, Generalization as an Extension, Generalization as a Restriction, Metadata, Constraints

---

**BLOCK 3 : UNIFIED MODELLING LANGUAGE-I**

---

**Unit 7      INTRODUCTION TO UML**

Introduction, The Importance of Modelling, Principles of Modelling, Object Oriented Modelling, Conceptual Model of the UML, Building Blocks of UML (Conceptual Model of UML), Things, Relationships, Diagram, Rules, Common Mechanisms in UML, Architecture

**Unit 8      BASIC AND ADVANCED STRUCTURE MODELLING**

Introduction, Classes, Relation, Common Mechanism, Diagrams, Advanced Classes, Advanced Relationship, Interfaces, Types and Roles, Packages

**Unit 9      CLASS & OBJECT DIAGRAM**

Introduction, Class Diagrams : Terms & Concepts, Modelling Techniques for Class Diagrams, Object Diagrams : Terms and Concepts, Modelling Techniques for Object Diagrams

**Unit 10     BASIC BEHAVIOURAL MODELLING-I**

Introduction, Basic Behavioural Modelling : Interactions, Interaction Diagrams

---

**BLOCK 4 : UNIFIED MODELLING LANGUAGE-II**

---

**Unit 11     BASIC BEHAVIORAL MODELING-II**

Introduction, Basic Behavioural Modelling : Use Case, Use Case Diagrams, Activity Diagrams



**Unit 12      ADVANCED BEHAVIORAL MODELING**

Introduction, Events and Signals, State Machines, Processes and Threads, Time and Space, State Chart Diagrams

**Unit 13      ARCHITECTURAL MODELING**

Introduction, Architectural Modelling : Component, Deployment, Component Diagrams, Deployment Diagrams

**Unit 14      CASE STUDY**

Introduction, UML Diagrams : Library Management System, Online Mobile Recharge



**Dr. Babasaheb Ambedkar  
Open University Ahmedabad**

**BCAR-502**

# **OBJECT ORIENTED ANALYSIS AND DESIGN**

---

## **BLOCK 1 : FUNDAMENTALS OF SYSTEM ANALYSIS AND DESING**

---

UNIT 1    SYSTEM ANALYSIS FUNDAMENTALS

UNIT 2    ANALYSIS MODELING

UNIT 3    SYSTEM DESIGN

# **FUNDAMENTALS OF SYSTEM ANALYSIS AND DESIGN**

## **Block Introduction :**

Object-oriented technology is very extensive and comprehensive. Users of computer systems have found that the effects of such technologies are increasingly user-friendly software applications and flexible operating systems operating in various industries such as banking, telecommunications, and television. The most important aspects of software development using object-oriented methodologies are object-oriented analysis, object-oriented design, and object-oriented implementation.

In this block, we will detail about System Analysis with types of system, Roll of system analyst, system development life cycle. The block will focus on the Analysis modelling in which we will detail about data flow approach, data dictionary, structured English, decision tables and decision tree.

In this block you will also learn and understand about the System Design which will focus on designing effective output, output design objective, data concepts, Normalization, Data warehouses.

## **Block Objectives :**

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

- About Object Oriented Analysis
- Basic of System Analysis
- Features of Analysis Modelling
- Concept of System Design

## **Block Structure :**

**Unit 1 : System Analysis Fundamentals**

**Unit 2 : Analysis Modelling**

**Unit 3 : System Design**

**UNIT STRUCTURE**

- 1.0 Learning Objectives
- 1.1 Introduction
- 1.2 Types of Systems
  - 1.2.1 Transaction Processing System (TPS)
  - 1.2.2 Management Information System (MIS)
  - 1.2.3 Decision Support System (DSS)
- 1.3 Role of System Analyst
- 1.4 System Development Life Cycle
- 1.5 CASE Tools
- 1.6 Interviewing
- 1.7 Let Us Sum Up
- 1.8 Answers for Check Your Progress
- 1.9 Glossary
- 1.10 Assignment
- 1.11 Activities
- 1.12 Case Study
- 1.13 Further Readings

**1.0 Learning Objectives :**

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

- Information is one the important resource of every organization.
- How to manage information ?
- How to manage computer generated information ?

**1.1 Introduction :**

Organizations have to manage the worker and the raw material. Information is one of the key resources of every organization. We have to understand that information is not only used for processing the business but it is also important to determine the success and failure of a business. If we want to make total use of information then a business is managed correctly. A manager should understand the cost of each phase of business.

**1.2 Types of Systems :**

Typical organizations are divided into operational, middle, and top levels. The information requirements for users at each level are different. To do this, there are many information systems that support all levels of the organization. You will learn about the different types of information systems, the organizational level at which they are used, and the characteristics of a particular information system.



**Figure 1.1 : Types of Systems**

### **1.2.1 Transaction Processing System (TPS) :**

Transaction Processing Systems are employed by organisations to keep track of daily business activities. Users at the operational management level use them. A transaction processing system's primary goal is to provide routine information, such as :

- How are printers sold today ?
- How much stock do we currently have ?
- How much is still owed to John Doe ?

The TPS system quickly responds to the queries above by keeping track of daily company transactions.

- Operational managers make routine, highly organised decisions.
- The transaction processing system generates extremely thorough information.

For instance, banks that offer loans demand that a person's employer have an MoU (memorandum of understanding) with the bank. The only thing the operational staff needs to do when a borrower whose employer has an MoU with the bank applies for a loan is to check the given paperwork. The documents for the loan application are processed if they satisfy the standards. The client is recommended to meet with tactical management staff to discuss the option of signing an MoU if they do not meet the standards.

Systems that process transactions include;

- Point-of-sale devices : track daily sales
- Systems for managing loans, processing staff salaries, etc.
- Systems for tracking inventory levels include : stock control systems;
- Airline reservation systems; and flight booking systems.

### **1.2.2 Management Information System (MIS) :**

Tactical managers utilise management information systems (MIS) to track the organization's present performance status. A management information system receives its input from a transaction processing system's output.

The tactical managers use the reports that the MIS system generates to monitor, regulate, and forecast future performance by routinely analysing the data with algorithms that aggregate, compare, and summarise the findings.

For instance, data from a point of sale system can be utilised to examine trends in the sales of both successful and unsuccessful products. Future inventory orders can be made using this information, increasing orders for products that are performing well and decreasing orders for underperforming products.

- Sales management systems, which receive data from the point-of-sale system, are examples of management information systems.
- Budgeting systems – provides an overview of how much money is spent both temporarily and permanently within the organisation.
- The human resource management system, which considers factors like employee turnover and general well-being.

The semi-structured decision is the tactical manager's responsibility. The tactical managers make judgement decisions based on their knowledge and the information provided by MIS systems to anticipate how much inventory or items should be ordered for the second quarter based on the first quarter's sales.

### **1.2.3 Decision Support System (DSS) :**

Senior management uses decision support tools to make complex decisions. Both internal (such as transaction processing systems and management information systems) and external systems provide input to decision support systems.

Decision support systems' primary goal is to offer unique, ever-changing challenges with solutions. Systems that support decision-making provide answers to problems like :

- How would doubling the factory's production lot affect employees' performance ?
- In the event that a new competitor entered the market, what would happen to our sales ?

Decision support systems are highly interactive and use complex mathematical models and statistical methods (probability, predictive modelling, etc.) to deliver solutions.

Examples of decision support systems include;

- **Financial planning systems** – It allow managers to assess alternate strategies for attaining objectives. Finding the best method to accomplish the goal is the goal. For instance, the formula  $\text{Total Sales less (Cost of Goods + Expenses)}$  is used to determine a company's net profit. Senior executives will be able to make adjustments to the values for total sales, cost of products, etc. using a financial planning system to examine the impact of decisions on net profit and determine the best course of action.
- **Bank loan management systems** – these are employed to assess the borrower's creditworthiness and estimate the possibility that the loan will be repaid.

#### **☐ Check Your Progress – 1 :**

1. Full form of TPS is \_\_\_\_\_
  - a. Transaction Processing System
  - b. Transmission Processing System
  - c. Transaction Product System
  - d. Transmission Product System

2. Full form of DSS is \_\_\_\_\_.
  - a. Decision Sensitive System
  - b. Decision Support System
  - c. Data Sensitive System
  - d. Data Support System
3. Full form of MIS is \_\_\_\_\_.
  - a. Management Integrated System
  - b. Mass Information System
  - c. Management Information System
  - d. Mass Integrated System

### **1.3 Role of System Analyst :**

An information technology (IT) expert with a focus on the analysis, design, and implementation of information systems is known as a systems analyst. Systems analysts coordinate with end users, software providers, and programmers to achieve these results by evaluating the suitability of information systems in terms of their intended outcomes. A systems analyst is a person who employs design and analysis methods to leverage information technology to address business challenges. Systems analysts can act as change agents by identifying the organisational adjustments that are required, designing the systems to carry out those adjustments, and motivating others to use the systems.

Despite the fact that they could be knowledgeable on a range of programming languages, operating systems, and computer hardware platforms, they typically stay out of the actual hardware or software development process. They might be in charge of creating schedules for implementation, personnel impact reduction, design considerations, and cost analyses.

A systems analyst frequently collaborates with a business analyst and is typically restricted to a certain system. Despite considerable overlap, these roles are not the same. In some cases, a business analyst will develop a solution without going too deeply into its technical details, relying instead on a systems analyst to do so. A business analyst will assess the business requirement, determine the best solution, and, to some extent, design a solution. A systems analyst will often evaluate code, review scripting and, possibly, even modify such to some extent.

The distinction between a business analyst and a systems analyst can be effectively blurred by some committed individuals who have practical experience in both fields (business analysis and systems analysis).

The System analyst examines input and process the data and gives output information. The System analyst support different business function through the use of computerized information system.

The basic goal of system analyst is to make analysis of the data imputed by the user according to the requirements of the user. The system analyst should experience and be able to work with people in all descriptions.

A system analyst has many roles to play such as an investigator, planner, designer, modulator, communicator, implementer, trainer, change agent, architect, psychologist, sales person, motivator, politician etc.

The following are the specific responsibilities of a system analyst :

- Analyse the existing system and make a requirement list by discussing with the users.

- Prepare a conceptual (logical) design for the system based on the requirements.
- Establish the boundaries of the system to use the inputs, outputs and interface.
- Define the functions to be performed and the parameters to measure performance.
- Find out the internal structures of the system and their dependencies.
- Prepare mathematical models to support the evaluation of system performance.
- Make alternative solutions and their weighted evaluation to choose the best.
- Decompose the system into various logical sub-systems to be integrated later.
- Participate in system development, testing, integration, and implementation.
- Associate with users, developers, and management for steering the project.
- Act as a leader in all the phases during the system development.
- Work as a change agent and catalyst for process development.
- Prepare the project plan and schedule for phase wise project completion.
- Determine the system reliability, availability, and quality.
- Prepare system development cost estimates and perform cost benefit analysis.

#### **1.4 System Development Life Cycle :**

The System development life cycle is used of analysis and design. The best system can be developed by using system development life cycle. Fig 1.2 shows how many phases are there in SDLC. Actually there are seven phases of SDLC, each phase is dividing discretely and different activities are repeated. The SDLC does not consist of any separate step or operation.

##### **(1) Determining Issues, Opportunities, and Goals :**

###### **Activity :**

- Interviewing user management,
- summarising the information learned,
- determining the project's scope, and
- recording the outcomes

###### **Output :**

- A feasibility report that defines the issue and provides objective summaries so management may decide whether to move forward with the planned project.

##### **(2) Determining the Need for Human Information :**

###### **Activity :**

- Questionnaires;
- Interviews;
- Sampling and investing hard data



## Object Oriented Analysis and Design

- Learn the who, what, where, when, how, and why of the current system
- Prototyping
- Observing how the decision maker behaves in their surroundings.

### Output :

- Analyst begins to grasp how consumers complete their tasks when dealing with a computer and how to improve the new system's utility and usability.
- The analyst should be fully informed on the people, objectives, data, and procedure involved as well as the business functions.



Figure 1.2 : Seven Stages of System Developmental Life Cycle

### (3) Analysing System Needs :

#### Activity :

- Create data flow diagrams
- Complete the data dictionary
- Analyse the structured decisions made
- Prepare and present the system proposal

#### Output :

- Recommendation on what, if anything, should be done

### (4) Designing the Recommended System :

#### Activity :

- Design procedures for data entry
- Design the human-computer interface
- Design system controls

- Design files and/or database
- Design backup procedures

**Output :**

- Model of the actual system

**(5) Developing and Documenting Software :**

**Activity :**

- System analyst works with programmers to develop any original software
- Works with users to develop effective documentation
- Programmers design, code, and remove syntactical errors from computer programs
- Document software with help files, procedure manuals, and Web sites with Frequently Asked Questions

**Output :**

- Computer programs
- System documentation

**(6) Testing and Maintaining the System :**

**Activity :**

- Test the information system
- System maintenance
- Maintenance documentation

**Output :**

- Problems, if any
- Updated programs
- Documentation

**(7) Implementing and Evaluating the System :**

**Activity :**

- Train users
- Analyst plans smooth conversion from old system to new system
- Review and evaluate system

**Output :**

- Trained personnel
- Installed system

**☐ Check Your Progress – 2 :**

1. System Development Life Cycle has \_\_\_\_\_ stages.  
a. Three            b. Five            c. Six            d. Seven
2. Which of the following is the second step of System Development Life Cycle (SDLC) ?  
a. Identifying problems opportunities and objektivites  
b. Determining information requirements  
c. Analysing system needs  
d. Implementing and evaluating system

3. Which of the following is the last step of System Development Life Cycle (SDLC) ?
  - a. Identifying problems opportunities and objectives
  - b. Determining information requirements
  - c. Analysing system needs
  - d. Implementing and evaluating system

### **1.5 CASE Tools :**

For creating organized, accurate, productive and complete information system the system analyst is using a tool called Computer Aided Software Engineering (CASE) tool to improve their routine work through the use of automated support. The organization produces and management practices many spread by using the CASE tools. The analyst must rely on case tools to increase productivity and communication.

**Analysts can get the following advantages by using the CASE tools :**

- Boosting analyst output,
- enhancing user–analyst communication,
- integrating life cycle activities,
- Accurately analysing maintenance modifications, and
- Using both capital and lowercase.

**CASE tools are used for :**

- Diagramming
- Computer display and report generation
- Analysis tools
- Central repository
- Document generation
- Code generation.

The organisation and management of software development on large, complicated projects that involve numerous software components and humans is a particular use of computer–aided software engineering. It enables everyone involved in a project–designers, programmers, testers, planners, and managers–to have a shared understanding of its status (stage wise). It makes a structured, checkpoint–based process more likely. The business plans, design specifications, comprehensive code specifications, coded units, test cases, test results, marketing strategies, and service plans for the project may also be stored in it or linked to it via document and programme libraries. It promotes code and design reuse, cutting down on time and expense and raising quality.

UML, the industry–standard language for describing, visualising, creating, and documenting software system components, serves as the foundation for many CASE technologies. By creating a "blueprint" for development, it streamlines the difficult software design process.

Before purchasing a CASE tool, it is important to assess the pros and cons of the various options.

Before making any decisions, the following inquiries should be addressed :

- Is there a balance between the benefits provided and the price ?
- Can the interactive nature of the existing development life cycle and the continual development brought on by task duplication be stopped by a CASE tool ?
- Will selecting a certain tool using the CASE tool enhance the entire development process ?

**CASE tools' characteristics include :**

The following qualities a CASE tool ought to have :

- (1) **Standard Methodology** : A CASE tool should support industry–standard modelling and software development approaches. UML is currently used by CASE tools.
- (2) **Flexibility** : A CASE tool needs to give the user alternatives and flexibility for editors and other tools.
- (3) **Strong Integration** : All phases of software development must be integrated with CASE tools. This means that if a model is changed, the code documentation and everything associated design must also reflect the change. As a result, this provides a structured environment for software creation.
- (4) **Testing Software Integration** : CASE tools ought to have interfaces for automated testing tools. This aids in testing software for regression and other purposes in dynamic environments.

❑ **Check Your Progress – 3 :**

1. What is the full form of CASE ?
  - a. Computer Aided Software Engineering
  - b. Computer Aided System Engineering
  - c. Component Aided Software Engineering
  - d. Component Aided System Engineering

## **1.6 Interviewing :**

The order of the interviews is intended to go from functionally higher levels of staff to lower levels. It is crucial to have management support, and this support should have been secured at the project planning stage. Management should let the users know about this commitment.

The analyst will make the best use of the time allocated by carefully preparing for the interview. Before the interview, the user is given written notice of the goals of the interview with enough specificity to enable user preparation. When multiple analysts are looking at connected business processes during interviews, double teaming should be set up. The interviewee's time is used as efficiently as possible by avoiding potential repetition in overlapping areas.

Planning interviews with the use of the Interview Action List is beneficial.

The interviews should be organised and managed using an Interview Checklist as a guide. The interviewee's relative level of responsibility will determine the questions to ask and the approach to take. Senior management, for instance, won't appreciate being asked for exhaustive lists of data pieces;

## **Object Oriented Analysis and Design**

these will need to be delivered by a different party. The analyst must therefore exercise competence in tailoring the interview to the knowledge level and outlook of the interviewee while still achieving the goals.

The rapid documentation of the interview, including hard copies of the prototype externals, and the interviewee's verification or amendment are crucial actions to take after the interview.

The latter activity is useful as a memory jog for any promised follow-up activities.

- **Interview Notes :**

Give a summary of the interview rather than a complete report, highlighting the following points :

- Important concepts learnt during the interview;
- Actions to be taken; and consensus gained on important decisions.
- Modifications to the project scope, confirmation of such modifications, modifications to the contractual obligations, and unanswered questions
- Assurances given by the client

**Guidelines for style of interview notes include the following :**

Use first and last names to identify people in the notes (eg. T. Nakrani).

- Include a clear name, the document number, and, if applicable, the page or section, if references to documents are made in the interview notes.
- Only information that will affect the specification, the design, or the implementation should be recorded in the interview notes. o References to potential changes in scope or contractual needs must be presented in a language that does not suggest the Consultants acceptance of the change (for example, "A licence for xxx software may be required" rather than "Consultant" ABC must acquire a license for xxx software".).

### **1.7 Let Us Sum Up :**

In this unit we learnt the some fundamental topics of system analysis like which are the different types of system, what are the roles and responsibilities of the system analyst, what are the different steps of software development life cycle. We have also learn the CASE tools which is useful for system analyst and Interviewing process which help to system analyst for system analysis process.

### **1.8 Answers for Check Your Progress :**

**Check Your Progress 1 :**

1 : a            2 : b            3 : c

**Check Your Progress 2 :**

1 : c            2 : b            3 : d)

**Check Your Progress 3 :**

1 : a

**1.9 Glossary :**

1. **CASE Tools** – For creating organized, accurate, productive and complete information system the system analyst is using a tool called Computer Aided Software Engineering (CASE) tool to improve their routine work through the use of automated support.
2. **System Analyst** – is an information technology (IT) professional who works for organization for analysing, designing and implementing information systems.
3. **SDLC** – It is process for developing any information systems.

**1.10 Assignment :**

1. How many stages are there in System Development Life Cycle ? Explain in detail.
2. What is CASE tool ?

**1.11 Activities :**

1. Study about MIS, DSS and TPS and differentiate these three.

**1.12 Case Study :**

1. Explain the Interviewing process in detail.

**1.13 Further Readings :**

1. Maitri Jhaveri, Madhuri B. Arhunshi – Structured and object oriented analysis and design methodology – person

**UNIT STRUCTURE**

- 2.0 Learning Objectives
- 2.1 Introduction
- 2.2 Data Flow Diagram
  - 2.2.1 History of the DFD
  - 2.2.2 Symbols and notations used in DFDs
  - 2.2.3 DFD Rules and Tips
  - 2.2.4 DFD Levels and Layers : From Context Diagrams to Pseudo Code
  - 2.2.5 Examples of DFD Use
  - 2.2.6 DFD and Unified Modeling Language (UML)
  - 2.2.7 Logical Vs. Physical DFD
- 2.3 Data Dictionary
  - 2.3.1 Types of Data Dictionaries
  - 2.3.2 Data Dictionary Components
  - 2.3.3 How to Create a Data Dictionary
  - 2.3.4 Pros and Cons of Data Dictionaries
- 2.4 Structured English
  - 2.4.1 Pseudocode
  - 2.4.2 Guidelines for Choosing the Right Tools
- 2.5 Decision table
  - 2.5.1 Components of a Decision Table
- 2.6 Decision Trees
- 2.7 Let Us Sum Up
- 2.8 Answers for Check Your Progress
- 2.9 Glossary
- 2.10 Assignment
- 2.11 Activities
- 2.12 Case Study
- 2.13 Further Readings

**2.0 Learning Objectives :**

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

- Understand the logical modelling of processes by studying examples of data flow diagrams.
- Draw data flow diagrams following specific rules and guidelines that lead to accurate well-structured process models.

- Decompose data flow diagrams into lower-level diagrams.
- Use data flow diagrams as a tool to support the analysis of information systems.

## **2.1 Introduction :**

The system analyst must be able to make use of data flow diagrams, which show graphical representation of data. The data flow diagram depicts the broadest possible overview of system inputs, processes and output, which corresponds to those of the general system. A layered data flow diagram may also be used to represent and analysed detail procedure within the larger system. This unit provides the information on DFD, how to draw DFD's, how to develop DFD's, logical and physical DFD's, use of Data dictionary, and how to create a data dictionary. Decision tables and decision trees, after successive levels of DFD, helps the system analyst catalogue the data processes , flows, stores, structures in data dictionary.

## **2.2 Data Flow Diagram :**

A data flow diagram (DFD) maps the flow of information for any process or system. It uses defined symbols such as rectangles, circles, and arrows, as well as short text labels to display data inputs and outputs, storage points, and routes between each destination. Data flow diagrams can range from simple summaries of processes, even hand-drawn, to detailed multi-level DFDs that gradually delve deeper into how data is handled. They can be used to analyse an existing system or model a new one. Like all the best charts and diagrams, a DFD can often visually "say" things that would be difficult to put into words and works for both technical and non-technical audiences, from developers to CEOs.

A data flow diagram (DFD) depicts how information moves through any system or process. It displays data inputs and outputs, storage locations, and routes between each destination using predefined symbols such rectangles, circles, and arrows as well as brief text labels. Data flow diagrams can range from straightforward, even hand-drawn summaries of operations to intricate, multi-level DFDs that gradually reveal more and more about how data is processed. They can be used to model a new system or analyse an existing one. A DFD, like the best charts and diagrams, can frequently "speak" things graphically that are challenging to express in words. It is appropriate for both technical and non-technical audiences, from developers to CEOs. Because of this, DFDs are still very common today. Today, they are less suitable for viewing interactive, real-time, or database-driven software or systems, even though they still function effectively for software and data flow systems.

### **2.2.1 History of the DFD :**

Data flow diagrams were popularized in the late 1970s, starting with the book *Structured Design* , by computer pioneers Ed Yourdon and Larry Constantine. They based it on David Martin and Gerald Estrin 's "data flow graph" calculation models. The structured design concept took off in the field of software engineering and the DFD method took off with it. It has become more popular in corporate circles, as it has been applied to business analysis, than in academic circles.



## **Object Oriented Analysis and Design**

### **Two related concepts also contributed :**

- Object Oriented Analysis and Design (OOAD), proposed by Yourdon and Peter Coad to analyze and design an application or system.
- Structured Systems Analysis and Design Method (SSADM), a waterfall method for analyzing and designing information systems. This rigorous approach to documentation contrasts with modern agile approaches like Scrum and the Dynamic Systems Development Method (DSDM).

Three other experts who contributed to this rise in DFD methodology were Tom DeMarco , Chris Gane , and Trish Sarson . They have come together in different combinations to be the main definers of the symbols and notations used for a data flow diagram.

### **2.2.2 Symbols and notations used in DFDs :**

Three common symbol systems are named after their creators :

- Yourdon and Coad
- Yourdon and DeMarco
- Gane and sarson

An important difference in their symbols is that Yourdon–Coad and Yourdon– DeMarco use circles for processes, while Gane and Sarson use rounded rectangles, sometimes called rhombuses. There are other variations of symbols in use as well, so the important thing to keep in mind is to be clear and consistent in the forms and notations you use to communicate and collaborate with others.

Using the DFD rules or the guidelines of any convention, the symbols describe the four components of data flow diagrams.

1. **External Entity :** An external system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of the information that enters or leaves the system. They can be an external organization or person, a computer system, or a business system. They are also known as terminators, springs, and wells or actors. They are usually drawn at the edges of the diagram.
2. **Process :** Any process that modifies data, producing an output. It could perform calculations or sort data based on logic or direct data flow based on business rules. A short tag is used to describe the process, for example "Send Payment".
3. **Data Store :** Files or repositories that contain information for later use, such as a database table or membership module. Each data store is given a simple label, for example "Orders".
4. **Data Flow :** The path that data travels between external entities, processes, and data stores. It represents the interface between the other components and is displayed with arrows, usually labeled with a short data name, such as "Billing Details".


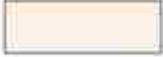






Notation	De Marco & Yourdon	Gane and Sarson
External Entity		
Process		
Data Store		
Data Flow		

Figure 2.1 : DFD Notations

**2.2.3 DFD Rules and Tips :**

1. Every process must have at least one input and one output.
2. Each data store must have at least one input and one output data stream.
3. Data stored in a system must go through a process.
4. All processes in a DFD go to another process or data store.

**2.2.4 DFD Levels and Layers : From Context Diagrams to Pseudo Code :**

A data flow diagram can progressively drill down into more detail using layers and layers, focusing on a particular piece. DFD levels are numbered 0, 1, or 2 and occasionally even go as high as level 3 or higher. The level of detail needed depends on the extent of what you are trying to accomplish.

- An alternative name for DFD level 0 is a context diagram. It is a fundamental summary of the entire system or process that is being studied or modelled. The system is represented as a single, high-level process, together with its relationship to external entities, in this fast view. A large audience, including stakeholders, business analysts, data analysts, and developers, should be able to understand it with ease.

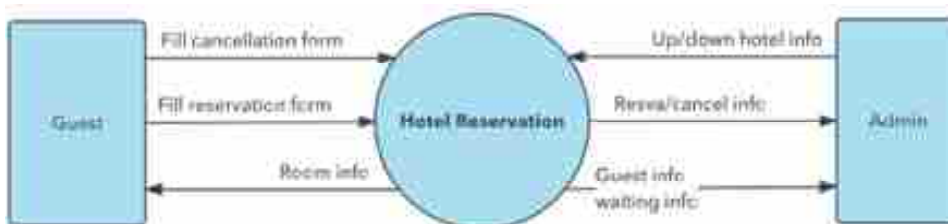
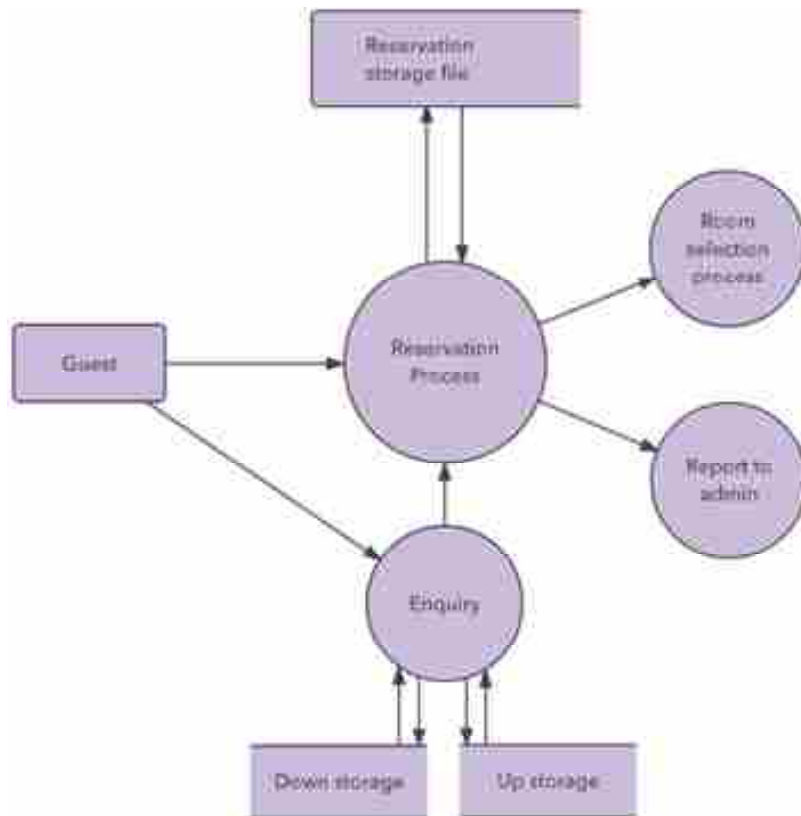


Figure 2.2 : DFD-0 Level

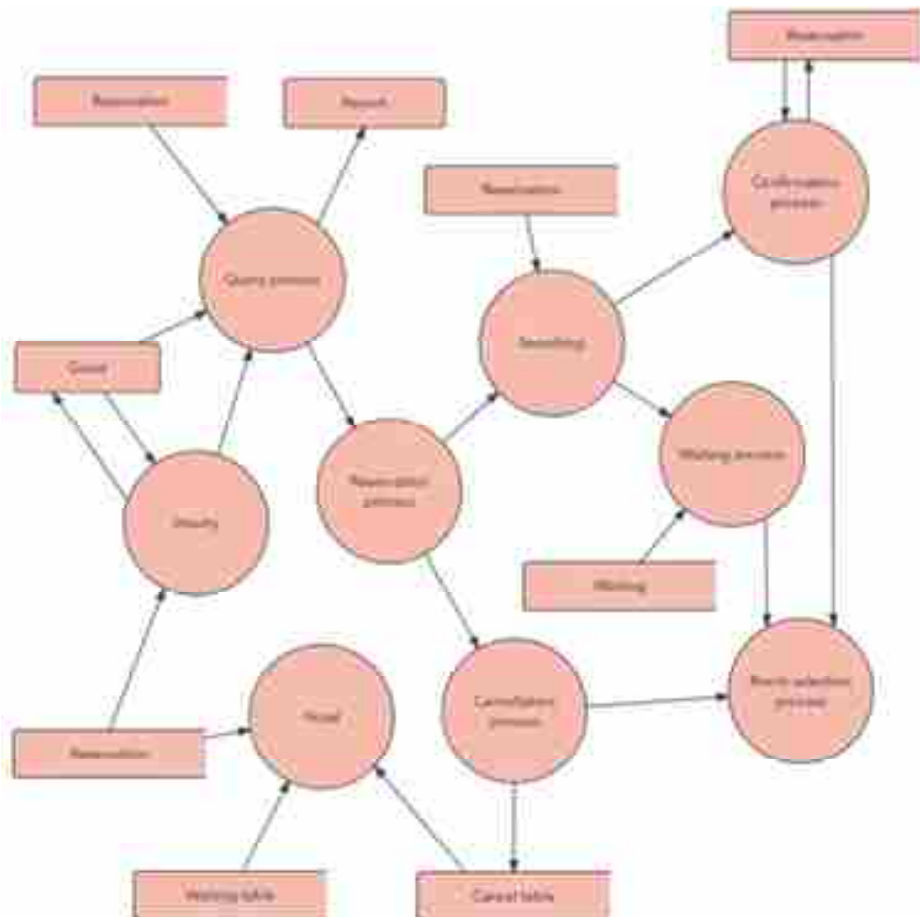
- DFD Level 1 provides a more detailed analysis of the diagram pieces at the context level. You will highlight the main functions performed by the system, breaking down the high-level process of the Context Diagram into its sub-processes.

**Object Oriented  
Analysis and Design**



**Figure 2.3 : DFD Level-1**

- Level 2 of the DFD then goes one step further into the parts of Level 1. More text may be needed to achieve the necessary level of detail on the operation of the system.



**Figure 2.4 : DFD Level-2**

- It is possible to go to levels 3, 4 and beyond, but going beyond level 3 is rare. This can create complexity that makes it difficult to communicate, compare, or model effectively.

With DFD levels, cascading levels can be nested directly on the diagram, providing a cleaner look with easy access to deeper dive.

By getting detailed enough in the DFD, developers and designers can use it to write pseudocode, which is a combination of English and a coding language. Pseudocode makes it easy to develop real code.

### **2.2.5 Examples of DFD use :**

Data flow diagrams are suitable for analyzing or modeling various types of systems in different fields.

**DFDs in software engineering :** This is where data flow diagrams got their main start in the 1970s. DFDs can provide a focused approach to technical development, where more research is done up front to get to coding.

**DFD in Business Analytics :** Business analysts use DFD to analyze existing systems and find inefficiencies. The process diagram can reveal steps that might otherwise be overlooked or not fully understood.

**DFDs in Business Process Reengineering :** DFDs can be used to model better and more efficient data flow through a business process. BPR was pioneered in the 1990s to help organizations reduce operating costs, improve customer service, and better compete in the marketplace.

**DFDs in Agile Development :** DFDs can be used to visualize and understand business and technical requirements and plan next steps. They can be a simple yet powerful tool for communication and collaboration to focus on rapid development.

**DFD in system structures :** Any system or process can be analyzed in progressive detail to improve it, both at a technical and non-technical level.

### **2.2.6 DFD and Unified Modeling Language (UML) :**

UML is a modelling language used in object-oriented software design to provide a more in-depth view, whereas a DFD shows how data flows through a system. A DFD can still serve as a solid starting point, but when creating the actual system, developers may use UML diagrams like class diagrams and structure diagrams to reach the necessary specificity.

### **2.2.7 Logical Vs. Physical DFD :**

These two groups make up a data flow diagram. A logical DFD depicts the data flow necessary for business operation. In contrast to how the system functions or is planned to function, it focuses on the activity and information required. A physical DFD, however, illustrates how the system is currently installed or will seem. For instance, the processes in a logical DFD would be business activities, whereas the processes in a physical DFD would be software and manual processes.

#### **☐ Check Your Progress – 1 :**

1. Which symbol is used for data flow ?
  - a. Arrow
  - b. Circle
  - c. Open ended box
  - d. Square

2. In DFD input/output is represented by \_\_\_\_\_
  - a. Rectangle
  - b. Circle
  - c. Open ended box
  - d. Square
3. \_\_\_\_\_ graphical notations is used to describe the data flow between various processes of a system ?
  - a. Entity relation diagram
  - b. Data analysis diagram
  - c. Data flow diagram
  - d. None of these

### **2.3 Data Dictionary :**

For the benefit of programmers and other users who may need to refer to it, a data dictionary is a collection of descriptions of the objects or data items in a data model. A centralised collection of metadata is frequently a data dictionary.

Identifying each interactive object in a system and their relationships to other objects is the first step in any analysis. Data modelling is the procedure involved in this. This creates an illustration of object relations. Each object or piece of data is given a descriptive name before its relationship is explained or included in a structure that does so implicitly. The default settings are listed, the data type, such as text, picture, or binary, is described, and a succinct textual description is given. This collection of data can be organized for reference in a book called a data dictionary.

A data dictionary can be used to understand where a data item fits into the structure, what values it can store, and what the data item actually signifies when creating programmes that use the data model. A bank or group of banks could, for instance, represent the data objects used in consumer banking. They could then provide bank programmers access to a data dictionary. Each data element in your consumer banking data model, such as "Account Holder" and "Available Credit," would be described in the data dictionary.

#### **2.3.1 Types of data dictionaries :**

Data dictionaries come in two different varieties. The degree of automatic synchronisation differs between active and passive data dictionaries.

- **Dictionary of active data :** These are internal data dictionaries that automatically describe and reflect any additions or modifications made to the host databases. This prevents any mismatch between the database architecture and the data dictionaries.
- **Dictionary resources for passive data :** These are data dictionaries designed to store data dictionary information and were developed as new databases independent of the databases they represent. Passive data dictionaries must be carefully handled to guarantee there are no errors and require an extra step to stay in sync with the databases they describe.

#### **2.3.2 Data dictionary components :**

The specifics of a data dictionary's contents can change. Generally speaking, these parts are different kinds of metadata that describe the data.

- Data element properties; data object listings (names and definitions) (such as data type, unique identifiers, size, null capacity , indexes, and optionality)
- Diagrams of entity–relationships (ERDs)
- Reference data; diagrams at the system level

- Invalid quality indicator codes and missing data
- Business laws (eg, to validate the quality of data and schema objects)

**2.3.3 How to Create a Data Dictionary :**

It is crucial to take into account all available data management tools, such as databases and spreadsheets, while making plans to create a data dictionary.

The majority of database management systems (DBMS) and information systems made with CASE tools come with integrated active data dictionaries. To produce a data dictionary from Access-based or Access-connected data, for instance, use the Analyzer tool for Microsoft Access, which analyses and documents databases.

If it is not possible to automatically generate a machine-readable data dictionary, it is suggested that you submit a data dictionary from a single source such as a spreadsheet.

Within Excel, .XLS or .XLSX spreadsheets can be transformed into data dictionaries. Online templates are useful for creating this type of data dictionary.

**2.3.4 Pros and Cons of Data Dictionaries :**

Data dictionaries can be a valuable tool for organizing and managing large lists of data. Other benefits include :

- Provides an organized and complete list of data.
- easy to search
- Can provide reporting and documentation for data across multiple programs
- Simplify the framework for system data requirements
- No data redundancy
- Maintains data integrity across multiple databases
- Provides information about the relationship between the different tables in the database.
- Useful in the software design process and in test cases

Although they provide comprehensive lists of data attributes, data dictionaries can be difficult for some users to use. Other disadvantages include :

- Functional details not provided
- not visually appealing
- Difficult to understand for non-technical users

**□ Check Your Progress – 2 :**

1. Data dictionary is also referred with other name i.e. \_\_\_\_\_  
 a. Function Catalog                      b. Data Catalog  
 c. System Catalog                        d. Storage Catalog
2. Metadata about relation are stored in \_\_\_\_\_  
 a. Function dictionary                    b. Data dictionary  
 c. Sequence dictionary                  d. Storage dictionary
3. \_\_\_\_\_ is a data about data.  
 a. Metadata      b. Grand data      c. Theta data      d. Gaeta data

## **2.4 Structured English :**

Structure English is derived from Structured Programming Language which provides a more understandable and accurate description of the process. It is based on procedural logic that uses constructive and imperative sentences intended to perform the operation for the action.

- It is best used when sequences and loops must be considered in a program and the problem requires action sequences with decisions.
- It does not have strict syntax rules. Express all logic in terms of iterations and sequential decision structures.

For example, see the following sequence of actions :

```
if the customer pays in advance
  then
    Give 10 % Discount
  else
    if purchase amount > = 25,000
      then
        if the customer is a regular customer
          then Give 10 % Discount
          else No Any Discount
        end i
      else No Any Discount
    end if
  end if
```

### **2.4.1 Pseudocode :**

A pseudocode does not conform to any programming language and expresses the logic in plain language.

- You can specify physical programming logic without actual coding during and after physical design.
- It is used in conjunction with structured programming.
- Replaces the flowcharts of a program.

### **2.4.2 Guidelines for Choosing the Right Tools :**

Use the following guidelines to select the most suitable tool for your needs :

- Use high-level or low-level analysis DFDs to provide good system documentation.
- Use the data dictionary to simplify the structure and meet the data requirements of the system.
- Use structured English if there are many loops and the actions are complex.
- Use decision tables when you need to check a large number of conditions and the logic is complex.
- Use decision trees when the sequence of conditions is important and there are few conditions to test.

❑ **Check Your Progress – 3 :**

1. What is the objective of using structure English from following ?
  - a. Ease reading programs for DFDs
  - b. Create algorithm from DFD.
  - c. Expanding the DFD for easily understand by the user.
  - d. Explain the computational procedures reasonably precisely which can be understood by any user

**2.5 Decision Table :**

Decision tables are a method to describe the complex logical relationship in a precise and easy to understand way.

- It is useful in situations where the resulting actions depend on the occurrence of one or more combinations of independent conditions.
- It is a matrix that contains rows or columns to define a problem and actions.

**2.5.1 Components of a Decision Table :**

- **Condition Stub** – Found in the upper left quadrant that lists all the conditions that need to be checked.
- **Action Stub** – Found in the lower left quadrant that describes all the actions that need to be taken to satisfy this condition.
- **Condition Entry** – Located in the upper right quadrant, it provides answers to the questions posed in the Condition Ticket quadrant.
- **Action Entry** – Found in the lower right quadrant that indicates the appropriate action that results from responses to the conditions in the condition input quadrant.

The entries in the decision table are given by the Decision Rules that define the relationships between the combinations of conditions and the lines of action. In the rules section,

- Y shows the existence of a condition.
- N represents the condition, which is not fulfilled.
- A blank space – against the action indicates that it should be ignored.
- X (or a check mark is ok) against the action indicates that it should be performed.

For example, see the following table :

**Table 1 : Decision Table**

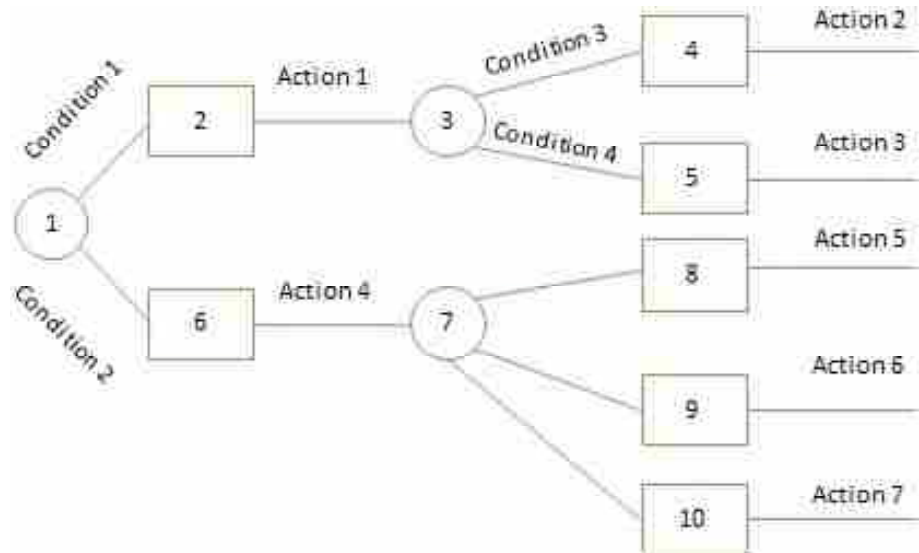
CONDITIONS	Rule 1	Rule 2	Rule 3	Rule 4
advance payment made	Y	N	N	N
Purchase amount = Rs 25,000/-	–	Y	Y	N
Regular customer	–	Y	N	–
Action				
Make a 10% discount.	X	X	–	–
do not discount	–	–	X	X



**2.6 Decision Trees :**

When explaining decisions and preventing communication issues, decision trees are a useful tool for defining complex relationships. An illustration of potential outcomes and conditions within a horizontal tree form is called a decision tree. This explains which circumstances should be taken into account first, second, and so forth.

Decision trees explain the connection between each condition and the possible responses. A circle denotes a condition, while a square denotes an action. It forces analysts to analyse the chain of decisions and identifies the actual decision that needs to be taken.



**Figure 2.5 : Decision Tree**

The fundamental drawback of a decision tree is that it doesn't provide enough information in its format to specify what other sets of circumstances it might test under. It offers a distinctive illustration of the connections between circumstances and behaviours.

See the following decision tree as an illustration :



**Figure 2.6 : Decision Tree Examples**

**❑ Check Your Progress – 4 :**

1. Difference between decision tree and decision table is :
  - a. One shows the logic and other shows the process
  - b. Form of representation
  - c. Value to end user
  - d. All of the above

2. Which of the following is/are used in decision tree ?
  - a. Branches and nodes
  - b. Graphical representation of alternate conditions
  - c. Consequences of various depicted alternatives
  - d. All of the above

### 2.7 Let Us Sum Up :

In this unit we understand the logical modelling of processes by studying examples of data flow diagrams following specific rules and guidelines that lead to accurate well-structured process models. We also learn that how to decompose data flow diagrams into lower-level diagrams.

We also learn different concepts like Data dictionary, Structured English, decision tables and decision trees which help to develop the logical model of some application development.

### 2.8 Answers for Check Your Progress :

- Check Your Progress 1 :**  
1 : a          2 : a          3 : c
- Check Your Progress 2 :**  
1 : c          2 : b          3 : a
- Check Your Progress 3 :**  
1 : d
- Check Your Progress 4 :**  
1 : b          2 : d

### 2.9 Glossary :

1. **Data Flow Diagram** – represents a system's major processes, data flows, and data stores at a high level of detail.
2. **Process** – The process of discovering discrepancies between two or more sets of data flow diagrams or discrepancies within a single DFD.
3. **Decision Table** – Decision tables are a method to describe the complex logical relationship in a precise and easy-to-understand way.
4. **Decision Tree** – A decision tree is a diagram showing alternative actions and conditions within the horizontal tree structure.
5. **Data Dictionary** – A data dictionary is a collection of descriptions of objects or data items in a data model for the benefit of programmers and others who need to refer to it.

### 2.10 Assignment :

1. Explain the rules for drawing good data flow diagrams.
2. Explain the main use of decision tree in system analysis.
3. Explain decision table in detail.

**2.11 Activities :**

Talk with a system analyst who works at an organization. Ask the analyst to show you a complete set of DFDs from a current project. Interview the analyst about his or her views about DFDs and their usefulness for analysis.

**2.12 Case Study :**

A local college raises purchase order or buying stationery items for its office use. Invoices are sent by the vendors and the payment is made by the college. A copy of the invoice and the payment details are filed by the accounts department. A report of the list of stationery items bought is sent monthly to the office in charge. Draw a level-0 DF diagram with a complete list of data flows, Data stores, processes and sources/sinks.

**2.13 Further Reading :**

1. Maitri Jhaveri, Madhuri B. Arhunshi – Structured and object oriented analysis and design methodology – person
2. Norman, Ronald – object oriented system analysis and design – prentice hall 1996

**UNIT STRUCTURE**

- 3.0 Learning Objectives
- 3.1 Introduction
- 3.2 Design of Effective Results
- 3.3 Output Design Objectives
- 3.4 Data Concepts and Normalization
- 3.5 Data Warehouses
- 3.6 Human–Computer Interaction
- 3.7 Query Design
- 3.8 Effective Coding
- 3.9 Let Us Sum UP
- 3.10 Answer for Check Your Progress
- 3.11 Glossary
- 3.12 Assignment
- 3.13 Activities
- 3.14 Case Study
- 3.15 Further Readings

**3.0 Learning Objectives :**

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

- The element of input design for forms, screens and web fill in forms.
- How to store the data
- System users, their interfaces with the computer
- Ensuring Quality of data input to the information system

**3.1 Introduction :**

Forms are used to present or collect information about a single item, such as a customer, product, or event. Forms can be used for both input and output purposes. Reports, on the other hand, are used to convey information about a collection of objects. Form and report design is a key ingredient for a successful system. Since the user often equates the quality of a system with the quality of its input and output methods, we can see that the process of designing forms and reports is a particularly important activity. Information can be collected and formatted in many ways. It is useful for system analysts to catch up and understand the design's do's and don'ts and the trade-off between different format options.

### **3.2 Design of Effective Results :**

Output is information provided to users through the information system via intranet, extranet or World Wide Web. Some data require extensive processing before they become appropriate output; other data is stored and when retrieved, they are considered as output with little or no processing. The output can take many forms : the traditional paper copy of paper reports and digital copy such as display screens, microforms and video and audio output. Users rely on the output to perform their tasks and often judge the profitability of the system solely on the basis of its output. To create the most usable output possible, the system analyst works closely with the user through an iterative process until the output is considered satisfactory.

The output is any useful information or data provided by the information system or decision support system to the user. The output can take virtually any shape, including print, display, audio, microform, CD-ROM or DVD, and web-based documents.

The systems analyst has six main goals when designing the output. They must design the output to serve the intended human and organizational purpose to suit the user, deliver the right amount of output, deliver it to the right place, deliver the output on time and select the correct output method.

It is important for the analyst to realize that the output content is related to the output method. The production of different technologies affects the users in different ways. Output technologies also differ in their speed, cost, portability, flexibility, availability, and storage and retrieval capabilities. All these factors must be taken into account when choosing between print, visual, audio, electronic or web-based output or a combination of these.

The presentation of the result may affect the users in their interpretation of it. Analysts and users should be aware of sources of bias. Analysts need to interact with users to design and customize the output; inform users about the potential for skew in the output; create flexible and modifiable results; and allows users to use multiple results to help verify the accuracy of a particular report.

Printed reports are designed using computer-aided design software tools that have shape design templates and drag-and-drop interfaces. The data dictionary serves as the source of the required data in each report.

It is important to design the output for user screens, especially for DSS and the Internet. Aesthetics and ease of use are crucial when creating well-designed outputs for monitors. It is important to produce prototype screens and web documents that encourage users to interact with them and make changes where desired.

### **3.3 Output Design Objectives :**

Because useful output is essential to ensure the use and acceptance of the information system, there are six goals that the system analyst seeks to achieve when designing output :

1. Design the output to serve its intended purpose.
2. Output design to suit the user.
3. Delivers the appropriate amount of output.

4. Make sure the output is where it is needed.
5. Make sure the box is on time.
6. Select the correct output method.

### **1. Design results to serve the intended purpose :**

Each excursion must have a purpose. In the analysis phase with determining information requirements, the system analyst finds out what purposes exist for the user and the organization. The output is then styled for these purposes.

You will have several options for delivering results simply because the app allows you to do so. Remember, however, the intentionality rule. If output is not functional, it does not need to be created because there are time and material costs associated with all output from the system.

### **2. Output design to suit the user :**

With a large information system serving many users for many different purposes, it is often difficult to customize the output. Based on interviews, observations, cost considerations, and perhaps prototypes, it will be possible to design outputs that address what many, if not all, users need and prefer.

In general, it is more convenient to create user-specific or customized results when designing a decision support system or other highly interactive applications, such as those that use the Internet as a platform. However, it is still possible to design results that suit a user's tasks and roles in the organization, which brings us to the next goal.

### **3. Delivers the right amount of output :**

Part of the task of designing outputs is to decide how much output is right for the users. A useful heuristic is that the system must provide what each person needs to complete their work. This answer is still far from a complete solution because it may be appropriate to first display a subset of this information and then allow the user to easily access additional information.

The problem of information overload is so prevalent that it is a cliché, but it is still a valid concern. No one benefits from being given too much information just to show the system's capabilities. Always keep the decision makers in mind. Often they do not need a large number of results, especially if there is an easy way to access more via a hyperlink or a drill-down function.

### **4. Make sure the output is where it is needed :**

Output is often produced in one place and then distributed to the user. The increase in personally accessible online output has reduced the distribution problem somewhat, but proper distribution remains an important goal for the systems analyst. To be used and usable, the output must be presented to the correct user. No matter how well designed the reports are, they are worthless if not viewed by the relevant decision makers.

### **5. Make sure the box is on time :**

One of the most common complaints from users is that they do not receive the information in time to make the necessary decisions. Although timing is not everything, it does play an important role in output. Many daily reports are required, some monthly only, some annual and some only exceptional. Using well-known web-based results can also alleviate some issues with the timing of the results distribution. Precise timing of output can be crucial to business operations.

**6. Select the correct output method :**

Choosing the right exit method for each user is another goal in exit design. Much of the output is now displayed on screens and users have the option to print it to their own printer. The analyst must recognize the pros and cons associated with choosing an exit method. The costs are different; for the user, there are also differences in data availability, flexibility, durability, distribution, storage and retrieval, portability, and overall impact. The choice of exit methods is not trivial, nor is it usually a matter of course.

**❑ Check Your Progress – 1 :**

1. Which of the following is not a measure of output design ?
  - a. Sell the user in current forms
  - b. Make sure the box is on time
  - c. Design output to serve a specific purpose
  - d. Select the efficient output method
2. \_\_\_\_\_ is not a trend with printers.
  - a. Quieter printers
  - b. Multiple fonts
  - c. Several operator interventions
  - d. More flexibility
3. \_\_\_\_\_ is not a form of electronic output from the following option.
  - a. fax
  - b. bulletin board messages
  - c. Email
  - d. Report

**3.4 Data Concepts and Normalization :**

**What is database design ?**

**Database** design is a collection of processes that facilitate the design, development, implementation and maintenance of the company's data management systems. Properly designed databases are easy to maintain, improve data consistency and are cost effective in terms of disk storage space. The database designer determines how data elements are mapped and what data is to be saved.

The main purposes of DBMS database design are to produce logical and physical design models of the proposed database system.

The logical model focuses on the data requirements and the data to be stored regardless of physical considerations. You are not worried about how the data will be stored or where it will be physically stored.

The physical data design model involves translating the logical database design of the database into physical media using hardware resources and software systems such as database management systems (DBMS).

**Why is database design important ?**

Helps produce database systems.

1. They meet the requirements of the users.
2. Has high performance.

The database design process in DBMS is essential for a high-performance database system.

**Requirements analysis :**

- **Planning** – These stages of database design concepts relate to planning the entire life cycle of the database. Takes into account the organization's information system strategy.
- **System Definition** – This phase defines the scope and limits of the proposed database system.

**Database design :**

- **Logical model** – This phase deals with the development of a database model based on the requirements. All design is on paper without physical implementations or DBMS-specific considerations.
- **Physical model** – This step implements the logical model of the database taking into account DBMS and physical implementation factors.

**Implementation :**

- **Data Conversion and Loading** – This phase of relational database design involves importing and converting data from the old system to the new database.
- **Testing** – This phase deals with the identification of errors in the newly implemented system. Checks the database in relation to the requirements specifications.

**Two types of database techniques**

1. Standardization
2. ER modeling

**Normalization :**

You may avoid redundant data and inconsistent data by normalising your data. Normalization comes in various forms.

The functions are normalised to remove duplication after being defined and having properties chosen. If a device satisfies a set of conditions for a specific normal shape that represents this information, it is said to be normalised. First, second, third, and fourth normal forms are all possible for units, and each has its own set of rules. You don't always adhere to these guidelines; sometimes you do.

The typical form guidelines are cumulative. In other words, in order for an entity to satisfy the requirements of the second normal form, the requirements of the first normal form must also be satisfied. The first, second, and third standard forms' rules must also be followed by an entity that abides by the fourth standard form's regulations.

A special event is referred to as an instance in the context of logical data modelling. A set of data values for each attribute related to a device constitutes an instance of that device.

**First normal form :**

If a relational entity only ever has one value and never has many recurring attributes, it satisfies the first normal form criteria. Repeating characteristics are various traits that are fundamentally the same, often known as a repeating group. Each property of a unit that satisfies the first normal form criteria is independent and distinct in meaning.



**Second normal form :**

If every attribute that isn't in the primary key provides a fact that depends on the entire key, the device is in a different normal state. When a fact about a subset of a composite key is a non-primary key characteristic, another normal form is violated. For instance, a warehouse unit may keep track of the quantity of particular parts kept at particular department stores.

**Third normal form :**

If each non-primary key feature of a device gives a fact that is independent of other non-key attributes and depends solely on the key, the device is said to be in third normal form.

When a non-primary property is a fact of another non-key attribute, there is a third normal form violation.

**Fourth normal form :**

If no instance has two or more independent facts about the entity with different values, the entity is in fourth normal form.

Take the unit EMPLOYEES, for instance. Both SKILL CODE and LANGUAGE CODE may be present for each instance of EMPLOYEE. An employee might speak numerous languages and has a variety of abilities. Employees and skills have one relationship, and employees and language have the other.

**❑ Check Your Progress – 2 :**

1. In which of the following forms does a function have no partial functional dependencies ?  
a. 1 NF            b. 2 NF            c. 2 NF            d. 4 NF
2. 4 NF is designed to handle \_\_\_\_\_  
a. Dependence on several values    b. join addiction  
c. transitive dependence            d. All these
3. If each non-key attribute is functionally dependent on the primary key, the relation will be in :  
a. 1 NF            b. 2 NF            c. 3 NF            d. 4 NF

**3.5 Data Warehouses :**

The relational database management system (RDBMS) design known as a "data warehouse" satisfies the needs of transaction processing systems. Any centralised data warehouse that can be searched for commercial purposes can be broadly characterised as it. It is a database that houses data meant to support the needs of decision-making. It is a collection of decision-supporting technologies designed to empower knowledge workers (directors, managers, and analysts) to reach ever-higher levels of decision-making. In order to help company leaders systematically organise, comprehend, and apply their information to make strategic decisions, data warehousing supports architectural frameworks.

The data warehouse environment includes an ETL (Extract, Transport and Load) solution, an OLAP (online analytical processing) engine, client analysis tools and other applications that handle the process of collecting information and delivering it to business users.

**What is a data warehouse ?**

A relational database called a data warehouse (DW) is made for query and analysis rather than transaction processing contains historical information collected from transaction data from single and numerous sources.

A data warehouse focuses on supporting decision makers with data modelling and analysis while delivering integrated historical data from across the company.

A data warehouse is a collection of information that is unique to the entire company, not simply a particular user base.

It is utilised for decision-making rather than routine tasks and transaction processing.

A data warehouse can be seen as a computer system with the following attributes :

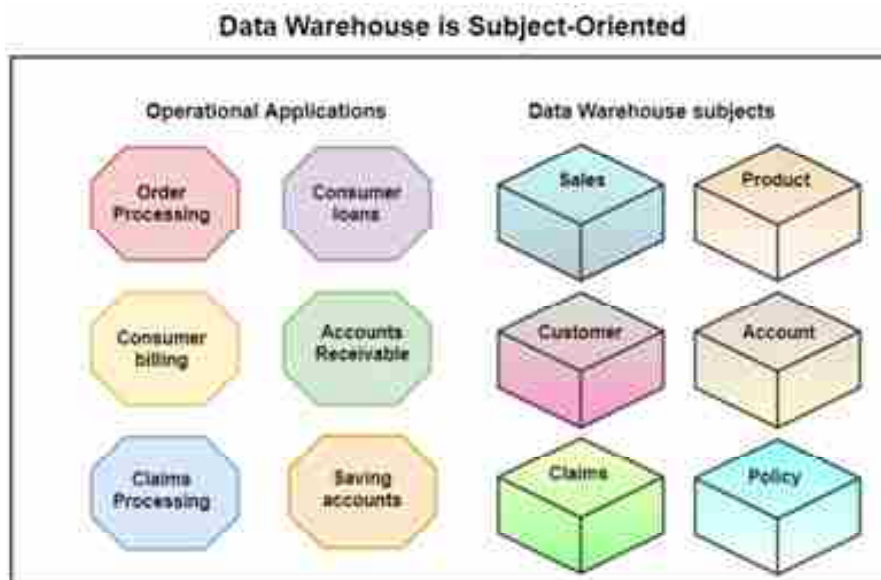
- It is a database with information from multiple programmes created for research assignments.
- Supports a limited number of customers who have lengthy conversations.
- It provides a historical perspective on the information by combining current and historical data.
- It is used for in-depth reading.
- Has a few substantial tables.

"Data Warehouse is a topic-oriented, integrated and time-varying repository of information to support management decisions."

**Data Warehouse features :**

**1. subject-oriented :**

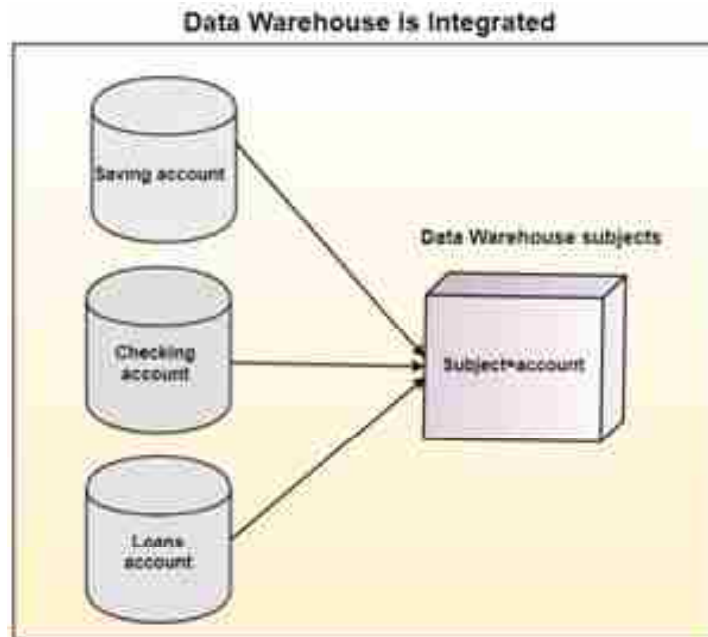
Data modelling and analysis for decision makers is an objective of a data warehouse. In contrast to the continuing operations of the entire business, data warehouses often provide a brief, direct overview of a specific issue, such as a customer, product, or sale. This is accomplished by removing information about the issue that is not helpful and including all the information users require to comprehend the topic.



**Figure 3.1 : The data warehouse is subject-oriented**

**2. Integrated :**

Various heterogeneous data sources, including RDBMS, flat files, and online transaction logs, are combined in a data warehouse. To maintain uniformity in name standards, attribute types, etc. between various data sources, data cleansing and integration must be performed during data storage.



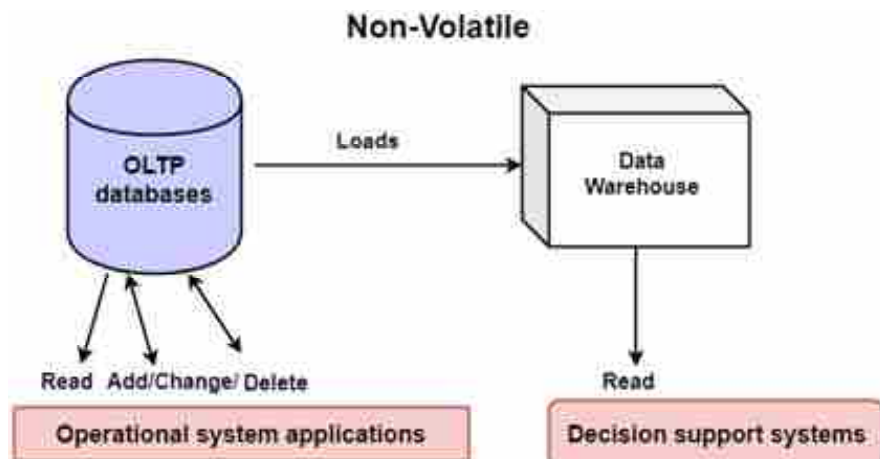
**Figure 3.2 : Data Warehouse is integrated**

**Time variant :**

In a data warehouse, historical data is kept. For instance, files from the last three, six, twelve, or even longer periods of time may be retrieved from a data warehouse. In a transaction system, only the most recent file is frequently stored, hence these differences.

**3. Non-volatile :**

The source's operational RDBMS is translated into the data store, which is a physically distinct data store. In other words, no updating, insertion, or deletion of operational data is done in the data warehouse. The initial data entry and the data access are often the only two steps needed to access the data. Because DW does not need concurrency, recovery, or transaction processing functions, data recovery can be significantly sped up. Nonvolatile refers to the idea that data shouldn't change after entering the store.



**Figure 3.3 : The data store is not volatile**

**Data Warehouse history :**

The idea of data warehousing was developed when IBM researchers Barry Devlin and Paul Murphy created the "Business Data Warehouse" in the late 1980s.

In its most basic form, data warehousing was created to enable an architectural model for the information flow between operating systems and decision support systems. The concept tries to address the several issues that flow raises, particularly its high cost.

Due to the lack of a data warehouse design, numerous decision support environments need a substantial amount of space. In large enterprises, it was common for several decision support environments to function independently.

**Data Warehousing targets :**

- To help inform and analyze
- Maintain historical information about the organization.
- Be the basis for decision making.

**Need for data warehouse :**

The data storage is necessary for the following reasons :



**Figure 3.4 : Need for data warehouse**

1. **Business user :** To examine historical summary data, business users need a data warehouse. These people don't know much about technology, therefore the information can be explained to them simply.
2. **Keep history data :** A data warehouse is needed to keep the varying historical data. This post is intended to be utilised in a variety of ways.
3. **Make strategic decisions :** The data in the data warehouse may be a factor in some plans. Consequently, the data warehouse influences strategic choice-making.
4. **for uniformity and consistency of data :** The user can effectively commit to the uniformity and consistency of the data by gathering data from many sources in one central location.
5. **Rapid response time :** The data warehouse must be able to handle loads and query types that are occasionally unexpected, which calls for a high degree of flexibility

Data Warehouse Benefits

1. Recognize market trends and improve your predictions.
2. Data warehouses are made to handle massive amounts of data well.
3. End users can traverse, comprehend, and query data warehouses more easily due to their structure.
4. Data warehouses may make it simpler to design and maintain queries that would be difficult to manage in numerous conventional databases.
5. Managing the need for a lot of information from many consumers can be done effectively utilising data warehousing.
6. Data warehousing allows for the analysis of a lot of historical data.

### **3.6 Human-Computer Interaction :**

Human-computer interaction, or HCI, is the study of how people use computers and how well they have (or have not) been designed to connect with people. Many larger businesses and academic institutions are currently researching HCI. With a few notable exceptions, computer system developers haven't historically given usability much thought. Many modern computer users would contend that manufacturers still don't focus enough on making their products "simple to use." The demand for the services that computers can give has always outpaced the necessity for usability, according to computer system developers, who contend that computers are extraordinarily difficult items to design and build.

Varied "cognitive styles," such as "left brain" and "right brain," and different ways of learning and sustaining knowledge and skills are major factors in HCI. Different users often create different views or mental models regarding their interactions. (people). Differences in nationality and culture can play a part. User interface technology is evolving quickly, offering new chances for interaction to which prior research findings might not be applicable. This is something to keep in mind when researching or developing HCI. User choices ultimately determine how quickly they learn to use new interfaces.

### **3.7 Query Design :**

The first step in building a query (or more often a set of queries) is a design step, based on identified requirements for answering the following critical query design questions :

1. What types of data sources should I consult ?
2. What is the structure of each data source; that is, how are the XML source schemas ?
3. What do I want the query result to look like (ie the query result) ? In other words, how do I structure the output ?
4. What should the target XML schema look like ? (The target form defines the structure of the query result.)
5. Which goal scheme design pattern should I use ?
6. What source conditions do I need to define to get the information I need from the data sources ? (Source conditions are links, concatenations, aggregations, etc. defined to filter the source data in a specific way).

Once you have designed or "modeled" your query in this way based on what you want the query to do, and defined a general strategy for filtering

information, you are ready to test your query. For queries that are not very simple, you will probably want to review, refine and test the query several times and add optimization if necessary.

**3.8 Effective Coding :**

One of the ways in which data can be entered more accurately and efficiently is through informed use of different codes. The process of putting ambiguous or cumbersome data in short digits or letters that are easy to enter is called coding (not to be confused with program coding). Encryption helps the system analyst to achieve the goal of efficiency because the data that is encrypted requires less time for people to enter and therefore reduces the number of items entered. Coding can also help with the correct classification of data at a later stage in the data transformation process.

In addition, encrypted data can save valuable memory and storage space. In short, coding is a way of being eloquent, yet concise in data capture.

In addition to providing accuracy and efficiency, codes must have a purpose that supports users. Specific types of codes allow us to process data in a specific way. Human purposes of coding include the following :

1. Track something.
2. Information classification.
3. Hide information.
4. revealing information.
5. Request the corresponding action.

**❑ Check Your Progress – 3 :**

1. data warehousing used in \_\_\_\_\_
  - a. Logical system
  - b. transaction system
  - c. Decision support system
  - d. All earlier
2. Which of the following is not the property data warehouse ?
  - a. Subject-oriented
  - b. Collection from Heterogeneous source
  - c. Time variant
  - d. Volatile
3. Which of the following actions can be performed on the data warehouse ?
  - a. Modify
  - b. Alter
  - c. Scanning
  - d. Read/write

**3.9 Let Us Sum Up :**

In this unit, we understand the input design element for forms, screens, and web-filling forms. Well design information should have efficiency, accuracy, ease of use , simplicity and consistency. Knowledge of many different design elements will enable the systems analyst to achieve this goal. The screen displays a cursor that continuously guides the user. The display must be kept simple, the display must be consistent and the display must be attractive. These are the general guidelines for a well-designed monitor.

How to store the data is important in the design of an information system. There are two approaches to storing data : the first is to store the data in

a single file and the second is to develop a database that can be shared by many users for a variety of applications as the need arises.

In this device, we also focus on the users of the system, their interfaces with the computer, their need for feedback. Ensuring the quality of data input to the information system is crucial for quality assurance. The quality of entered data can be improved by achieving three main objectives for data entry : efficient coding, efficient data capture and data validation.

### **3.10 Answers for Check Your Progress :**

- ❑ **Check Your Progress 1 :**  
1 : a            2 : c            3 : d
- ❑ **Check Your Progress 2 :**  
1 : b            2 : a            3 : b
- ❑ **Check Your Progress 1 :**  
1 : c            2 : d            3 : c

### **3.11 Glossary :**

1. **Effective Coding :** It is one of the means by which data can be entered more efficiently and accurately.
2. **Data Warehouse :** It is a relational database management system (RDBMS) design that meets the requirements of transaction processing systems .
3. **Normalization :** helps avoid redundancy and inconsistencies in data.

### **3.12 Assignment :**

1. List guidelines for good form design.
2. Explain objectives of designing user interfaces.

### **3.13 Activities :**

1. Prepare the guidelines for good form design and screen design.

### **3.14 Case Study :**

1. Discuss the design objective for input form, input screen or web based fill in forma

### **3.15 Further Readings :**

- 1 Maitri Jhaveri, Madhuri B. Arhunshi – Structured and object oriented analysis and design methodology –person
2. Norman,Ronald– object oriented system analysis and design –prentice hall 1996
3. Coad. P and Yourdon .E – "Object Oriented Design" – Yourdon press

## **BLOCK SUMMARY :**

In this block, you have learnt and understand about the System analysis, types of system, Roll of system analyst, system development life cycle. The block also focuses on Analysis modelling in that you have learnt in detail about data flow approach, data dictionary, structured English, decision tables and decision tree.

In this block you have also learnt and understand about the System Design which focused on designing effective output, output design objective, data concepts, Normalization, Data warehouses, Human-computer Interaction, Designing Queries and Effective Coding.

## **BLOCK ASSIGNMENT :**

### ❖ **Short Questions :**

1. What is the Object-Oriented Approach ?
2. What is MIS ?
3. What is DSS ?
4. What is Meta data ?
5. What is data dictionary ?

### ❖ **Long Questions :**

1. Distinguish between MIS, TPS & DSS.
2. Differentiate between logical & physical DFD.
3. Explain decision table & decision tree in detail.
4. List the guideline for good form design.



**Object Oriented  
Analysis and Design**

❖ **Enrolment No. :**

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

Unit No.	1	2	3
No. of Hrs.			

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

Items	Excellent	Very Good	Good	Poor	Give specific example if any
Presentation Quality	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Language and Style	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Illustration used (Diagram, tables etc)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Conceptual Clarity	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Check your progress Quest	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Feed back to CYP Question	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____

3. Any other Comments

.....

.....

.....

.....

.....

.....

.....

.....



**Dr. Babasaheb Ambedkar  
Open University Ahmedabad**

**BCAR-502**

# **OBJECT ORIENTED ANALYSIS AND DESIGN**

---

## **BLOCK 2 : BASICS OF OBJECT ORIENTED MODELLING**

---

UNIT 4 INTRODUCTION TO OBJECT ORIENTED MODELLING

UNIT 5 IMPORTANCE OF MODELING

UNIT 6 ADVANCED MODELING CONCEPTS

# ***BASICS OF OBJECT ORIENTED MODELLING***

## **Block Introduction :**

A model is an abstraction created to help you comprehend something before it is built. A model is simpler to manipulate than the original gadget since it leaves out non-essential information. A fundamental human ability that enables us to manage complexity is abstraction. Building models to test designs before they are completed has been a practise among engineers, artists, and craftspeople for thousands of years. This also applies to the creation of hardware and software systems. To create complex systems, a developer must abstract many system viewpoints, create models with exact notations, confirm that the models satisfy the system's requirements, then gradually add details to the models to create an implementation.

In this section, we'll go into more detail regarding the qualities of object-oriented modelling as a cutting-edge thinking tool for problem-solving with understanding of visualisation in the actual world. We'll also emphasise the value of modelling. The study of generalisation and hereditary abstractions in respect to the structure, behaviour, and interactions between classes will be the main topics of this block. It will give a general notion of metadata for data or for data that describes other data. You will learn and comprehend the fundamentals of the inheritance process in terms of object properties and the usability of information in this block. The idea of generalisation and its shared characteristics, represented as classes and super classes. Common traits relating to qualities, connections, or techniques are well described for you.

## **Block Objectives :**

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

- About object-oriented modeling functions
- On the benefits of object-oriented modeling
- The importance of modeling
- About qualities of class and objects
- About qualities of links and associations
- About the object and the class
- About multiple inheritance
- On generalization as a limitation

**Block Structure :**

**Unit 4 : Introduction to Object Oriented Modeling**

**Unit 5 : Importance of Modelling**

**Unit 6 : Advanced Modelling Concepts**

**UNIT STRUCTURE**

- 4.0 Learning Objective
- 4.1 Introduction
- 4.2 Object Oriented Modelling
- 4.3 Characteristics of Object Oriented Modelling
- 4.4 Links and Association
- 4.5 Generalization and Inheritance
- 4.6 An Object Model
- 4.7 Benefits of OO Modelling
- 4.8 Let Us Sum Up
- 4.9 Answers for Check Your Progress
- 4.10 Glossary
- 4.11 Assignment
- 4.12 Activities
- 4.13 Case Study
- 4.14 Further Readings

**4.0 Learning Objectives**

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

- Object-oriented modelling fundamentals
- Defining objects and classes
- Describe the ideas of relationships and linkages.
- Describe the advantages of object-oriented modelling;
- Describe the concepts of generalisation and inheritance.

**4.1 Introduction :**

The field of object-oriented technology is quite vast and deep. Users of computer systems noticed the effects of such technologies in the form of operating systems that are versatile and used by many different industries, including banking, telecommunications, and television, as well as software programmes that have become easier to use. Such object-oriented technology may, among other things, cover object-oriented project management, computer hardware, and computer-aided software technology in the context of software technology alone. A combination of abstract techniques makes up an interface. A class can inherit abstract interface shapes and implement interfaces. Although it isn't a class, writing it generally is the same as writing a class. A class's self-description as the characteristics and actions of an object is obvious to all. A class implements the behaviours from an interface.

## **4.2 Object Oriented Modeling :**

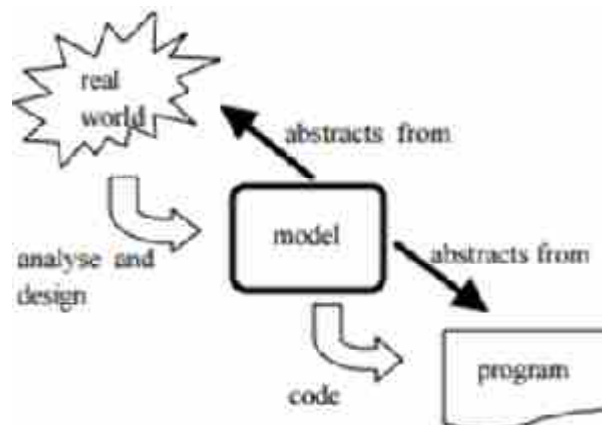
By offering sophisticated concepts, languages, methodologies, and tools to fit any or all stakeholders involved, the software engineering discipline supports the difficult and consequently error-prone work of software development. The usage of a software development process model, where the software development task is primarily separated into a number of specific sub-tasks, is an essential and now accepted method within software engineering. The creation of a system model is a crucial component of this step-by-step strategy. Such a model serves as an abstraction and outlines the conditions under which the software system can be created.

The process of creating objects using object-oriented modelling (OOM) involves grouping together objects that have instance variables storing values. Models that are record-oriented contrast with object-oriented values, which only contain objects.

By combining application and database development, the object-oriented modelling method creates a single data model and language environment. In addition to allowing data abstraction, inheritance, and encapsulation, object-oriented modelling enables object identification and communication.

The process of planning and creating the actual structure of the model's code is known as object-oriented modelling. A language that supports the object-oriented programming model is used to incorporate modelling approaches during the construction or programming phase.

Through three stages—analysis, design, and implementation—OOM develops object representation throughout time. Because the exterior components of the system are the main emphasis in the early stages of development, the model created is abstract. As the model develops, the primary emphasis moves from understanding how the system will be built to understanding how it should operate.



**Figure 4.1 : Representation of the model**

The early 1990s saw a decline in the use of object-oriented modelling techniques as more than 50 different approaches all professed to be the best. Through a commercial initiative that led to the creation of the object-oriented modelling language UML, which has been industrially standardised, this so-called technical war was put to an end.

❑ **Check Your Progress – 1 :**

1. Object-oriented modeling strategy means \_\_\_\_\_
  - a. use of objects
  - b. Use of algorithms
  - c. use of classes
  - d. a & c both

**4.3 Characteristics Object Oriented Modeling: Class and Objects :**

Object-oriented modeling is a whole new way of thinking about problems. This method is about visualizing elements through the use of models organized around ideas from the real world. Object-oriented models help understand problems, communicate with remote experts, modeling companies, and design programs and databases.

In object-oriented modeling, objects and their properties are described. In any system, objects arise to play some role. The object role definition method uses some object-oriented options.

➤ **Class and Objects :**

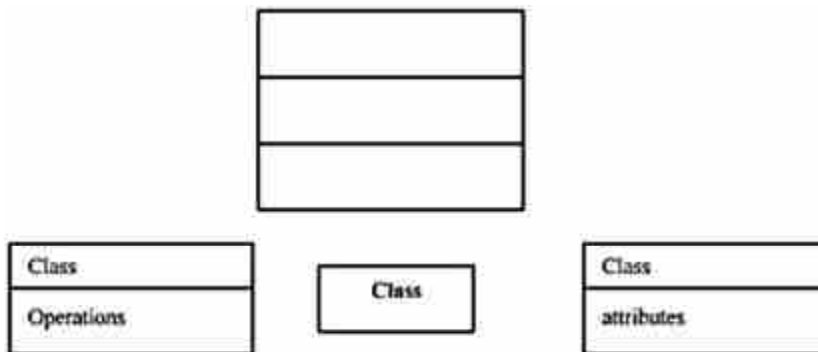
A class is a grouping of items or concepts with similar properties. These objects or ideas each have a name for an object. Classes specify the fundamental terms in the modelled system. The understanding and agreement of terms' meanings and other properties of system objects are generally substantially facilitated when a set of classes is used as the project's primary vocabulary.

Data modelling can be based on classes. Classes in OOM are occasionally the foundation of visual modelling tools like Rational Rose XDE, Visual Paradigm, function, and system model design.

A class can be a pattern, template, or model for a group of items with similar structural properties. Instances are objects made using the class. This is commonly known as the "class" or "cookie cutter" viewpoint. The cases are "cookies," as you would have guessed.

A class could be something that has both a mechanism for producing items that are compliant with the pattern and a pattern itself. Instances are the discrete elements that are "created" using the class generation method; this is the "class as an instance factory" viewpoint.

A class is a collection of all elements produced according to a specific pattern. If not, the class is the collection of all occurrences of that pattern.



**Figure 4.2 : Class performances**

A class is a group of objects with similar defining characteristics and shared behaviours. It provides a description or blueprint for the items that can be made with it. The process of creating an object that belongs to a class is known as instantiation. Consequently, an object is a class instance.



## Object Oriented Analysis and Design

Those who make up a class are

- A collection of properties for the objects that will be created from the class. Typically, the values of the properties vary to some extent among distinct objects belonging to the same class. Frequently, attributes are referred to as class data.
- A group of actions that represent how the class's objects behave. Functions and methods are other names for operations.

### ➤ **Example :**

Let us consider a simple class, Circle, that represents the geometrical figure circle in a two-dimensional space. The attributes of this class can be identified as follows ?

- x-coord, to denote x-coordinate of the center
- y-coord, to denote y-coordinate of the center
- a, to denote the radius of the circle

Some of its operations can be defined as follows ?

- findArea(), method to calculate area
- findCircumference(), method to calculate circumference
- scale(), method to increase or decrease the radius

Consider the straightforward class Circle, which depicts the geometric shape of a circle in two dimensions. The following are the characteristics of this class:

- a, to denote the radius of the circle;
- x-coord, to denote the x-coordinate of the centre;
- y-coord, to denote the y-coordinate of the centre

Its operations include some of the following:

- findArea() and findCircumference() are methods for calculating area and circumference, respectively.
- scale(), which raises or lowers the radius

At least some of the attributes have values assigned at the time of instantiation. To represent the state of an object like my circle, we can give it values like x-coord: 2, y-coord: 3, and a: 4. The value of the variable a will change to 8 if the operation scale() is applied to my circle with a scaling factor of 2. My circle's state has changed as a result of this action; it is now acting in a particular way.

### ➤ **Objects :**

Objects are physical as they are present in the universe around us. It has been found that the hardware; software, documents and concepts are all examples of objects. Using models, an officer sees personnel, buildings, departments, documents, and benefit packages as related objects. A person working on cars will see tires, doors, engines, speed and fuel level in the form of objects, while atoms, molecules, volumes and temperatures are objects according to the chemist, which can be perceived as a simulation of an object-oriented chemical. reaction.

It is usually considered that objects can be considered as a particular state. The state of an object is basically the state of an object or a set of circumstances that describes the object. It is generally seen that people talk about state information that is particularly related to a particular object. It is noted that the bank statement item will cover the most recent and available balance, which shows the condition of the watch item, which is available at the required time, showing the condition of the bulb that could be turned on or off. For complex objects such as a human or a car, a complete state description can be very complex. Fortunately, when we use objects to model the real world or imaginary things, we tend to generally limit the possible states of the objects to only those people who are relevant to our models.

In an object-oriented context, an object is a real-world component that may exist physically or conceptually. Each thing has

- A unique identity that sets it apart from other system items.
- A state that establishes an object's defining characteristics as well as the values of the properties the object possesses.
- Behaviour that reflects actions taken by an item in terms of modifications to its state that are externally visible.

The needs of the application might guide the modelling of the objects. A project, a process, a customer, a car, etc. are examples of objects that can have a physical presence or an ethereal intellectual life.

**□ Check Your Progress – 2 :**

1. Class means \_\_\_\_\_
  - a. Group of information
  - b. Collection of different types of objects.
  - c. Collection of similar items.
  - d. None of these

**4.4 Links and Associations :**

Links and association are ways of creating links that exist between objects and classes. It is observed that both the connections and the association have similar characteristics, while the connections are established between objects, while the connection is established between classes.

**Links :** In object modelling, links show how two items are related to one another. These objects or instances could differ or be identical in terms of behaviour and data structure. Thus, a link is a conceptual or physical relationship between two instances (or objects).

**Associations** are what object modelling refers to as a collection of compounds having a regular semantics and shared structure. The associations between the same classes are variations on all links between objects. The connection between classes is known as association.

1. The association
2. Relationship with the reverse address
3. The relationship between students and universities.

## Object Oriented Analysis and Design

The degree of affiliation is :

1. Unary association (degree of one)
2. Binary association (degree of two)
3. Ternary association (degree of three)
4. Quaternary Association (degree of four)
5. Higher order association (more than four)

### Check Your Progress – 3 :

1. Link in object modeling means \_\_\_\_\_
  - a. Physical or conceptual connection between instances
  - b. relationship between objects
  - c. the relationship between the classes
  - d. None of these

### 4.5 Generalization and Inheritance :

Powerful abstractions to categorise structure and/or behaviour into one or more classes include generalisation and inheritance. In general, a class's relationship to itself defines an abstraction hierarchy in which one or more subclasses derive from one or more superclasses. A triangle representing generalisation connects a superclass and its subclasses. A line runs from the top of the triangle to the superclass. Lines join the subcategories to a horizontal bar that is fastened to the base of the triangle.

A generalisation connection in UML modelling is a relationship that applies the idea of object orientation known as inheritance. When two things or objects have a generalisation relationship, one of them acts as the parent and the other as the kid. The parent's functionality is passed down to the child, who has access to and control over it. In class, component, deployment, and use case diagrams, generalisation relationships are used to indicate that the kid inherits the actions, traits, and relationships of its parent.

The generalisation relationship must employ the same types of model elements as other relationships in order to comply with UML's standards. For example, a generalisation relationship cannot be used between actors and use cases.

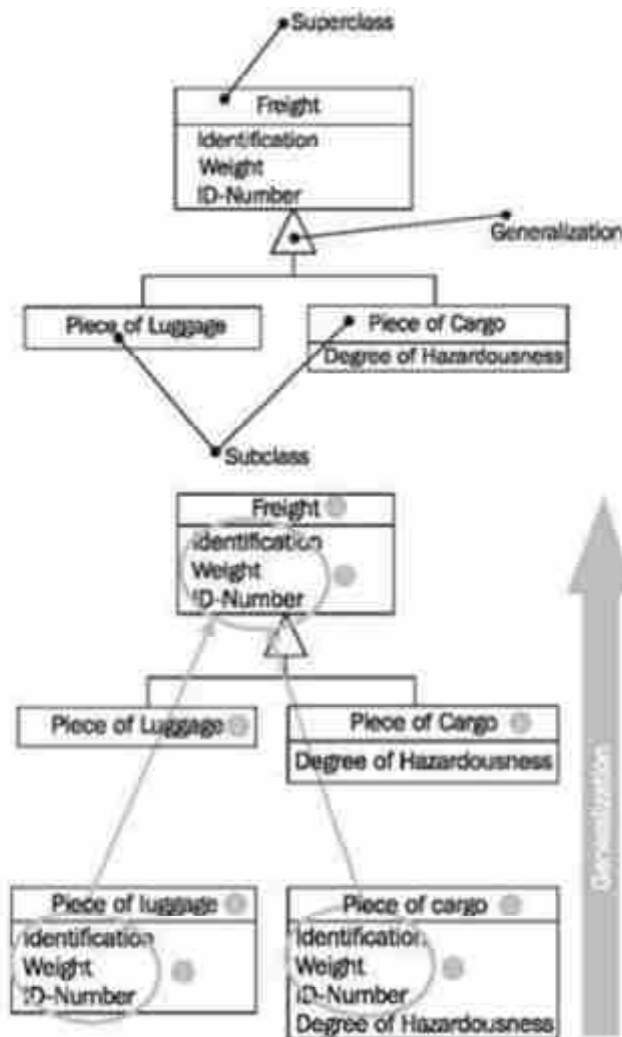


Figure 4.3 : Generalization Concepts

The reuse of code is referred to as inheritance in object-oriented development. We examine the resultant classes after modelling is complete and work to group comparable classes so that the code can be used. Inheritance, specialisation, and generalisation are all closely related. When discussing the relationship between classes, the term "generalisation" is used, while the term "inheritance" is used to share properties and operations using the generalisation relationship.

The ability to construct an abstract "contract" or "protocol" that a collection of concrete classes must adhere to is a feature shared by generalisation and inheritance. This enables you to design more compact and abstract code, which in turn enables code reuse, handling several comparable classes in a more consistent manner. Using the following as an illustration of a class implementation interface, one interface extends another interface:

```
interface Printable1 {
    void print ();
}

interface Showable1 extends Printable1 {
    void show ();
}
```

```
class Testinterface2 implements Showable1 {  
    public void print1 () {  
        System.out.println ( " Tulsidas ");  
    }  
  
    public void show1 () {  
        System.out.println ( " Nakrani ");  
    }  
  
    public static void main(String args []){  
        Testinterface2 obj = ny Testinterface2( );  
        obj.print1 ( );  
        obj.show1 ( );  
    }  
}
```

If we run this program we get an output such as:

Tulsidas  
Nakrani

❑ **Check Your Progress – 4 :**

1. Inheritance means \_\_\_\_\_
  - a. Reuse of code in object modeling
  - b. Get properties for one class in another class
  - c. Reuse of code in object modeling
  - d. All these

**4.6 An Object Model :**

A logical interface, piece of software, or system that is modelled using object-oriented methods is called an object model. Prior to development or programming, it permits the establishment of an architectural software or system model.

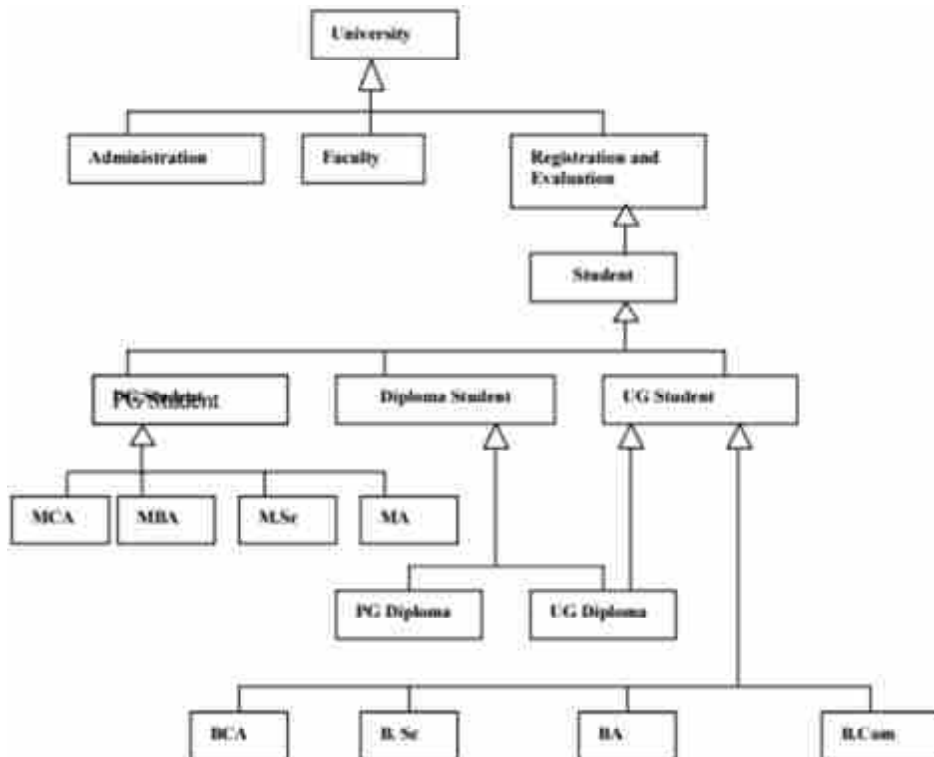
The object-oriented programming (OOP) lifecycle includes an object model.

A programme or system's objects and classes are defined or described by an object model. It defines inheritance, encapsulation, and other object-oriented interfaces and features, as well as the interfaces or interactions between various models.

The Document Object Model (DOM), a collection of objects that gives a modelled representation of dynamic HTML and XHTML-based Web pages, is an example of an object model.

The Component Object Model (COM) is a Microsoft-exclusive software architecture for building software components.

The object model visualizes all elements of a software application in the form of objects. The example object model is shown in Figure 4.4



**Figure 4.4 : Object model**

- An object's instance variables include values that are stored there.
- Unlike models that break records, these values are things in and of themselves.
- As a result, objects can embed other objects up to an arbitrary depth.
- Code bodies that interact with an object can also be found in objects.
- These bodies of code are referred to as methods.
- Classes are collections of objects with similar value types and methods.
- A class can be thought of as an object's type specification.
- Comparison: The idea of an abstract data type in programming languages.
- An object can only access the data of another object by calling that object's method.

➤ **Benefits of Object Model :**

After going through the fundamental ideas of object orientation, it would be beneficial to mention the benefits that this model has to offer.

The use of the object model has the following advantages :

- It speeds up software development.
- It is simple to keep up. If a module were to develop a bug, a programmer would be able to correct it while the rest of the software was still functioning.
- It allows for upgrades that are largely painless.
- It permits the reuse of components, layouts, and operations.
- It lowers the risks associated with development, especially when integrating complex systems.

❑ **Check Your Progress – 5 :**

1. Send message to object means \_\_\_\_\_
  - a. Send parameters to the object
  - b. Invoke the method of another object
  - c. Send parameters to the object
  - d. All these

**4.7 Benefits of OO Modeling :**

After going through the fundamental ideas of object orientation, it would be beneficial to mention the benefits that this model has to offer.

The following are some advantages of the object-oriented approach:

**Reduced maintenance :** The fundamental objective of object-oriented programming is to make sure that the system may live longer and require much less upkeep. Behaviors are also recycled and incorporated into new behaviours as a result of encapsulating the majority of system processes.

**Real-world modelling :** Compared to conventional techniques, the object-oriented approach has a tendency to model the real world considerably more thoroughly. Classes of objects are used to categorise items, and behaviour is connected to objects. Instead than being based on data and processing, the model is built on objects.

**Increased flexibility and reliability :** Because new behaviours are "constructed" from existing objects, object-oriented systems promise to be significantly more dependable than conventional systems. New objects can also be produced at any time because objects are dynamically called and accessible. Data properties from one or more other objects may be inherited by new objects. Without altering the operation of current systems, new behaviours are added as well as behaviours that are inherited from super classes.

**High code reuse :** When a new object is created, it immediately inherits the traits and data attributes of the originating class. Data and behaviour from any superclasses in which the new object participates will be inherited by it. A new object behaves "wigitty" despite having new behaviour defined in the system when a user creates a new widget type.

❑ **Check Your Progress – 6 :**

1. What are the benefits of object-oriented modeling from the following ?
  - a. Improve reliability
  - b. improve flexibility
  - c. reduce maintenance
  - d. All these

**4.8 Let Us Sum Up :**

In this unit, we have learned that object-oriented technology is broad and comprehensive, enhancing user-friendly software applications with flexible operating systems. You see, object-oriented modeling is a whole new way of thinking about problems, where you can visualize things by modeling those that are organized around real-world ideas.

As an instance factory, the class may be thought of as an instance of each individual element created by the class creation mechanism. The class is known to consist of a pattern and a mechanism for constructing elements

that support a pattern. We have looked into how associations and bonds may be used to create links between objects and classes. Since associations and bonds have many characteristics, they can be used to create bonds between objects and the associations that create classes.

Generalization and inheritance are strong abstractions to share class structure and behaviour and have a relationship between classes and exhibit a branch of abstraction where subclasses inherit from super classes. The challenge is defined with the identification of user needs and the construction of models in actual objects during the particular phase known as object-oriented analysis.

#### **4.9 Answers for Check Your Progress :**

**Check Your Progress 1 :**

1 : d

**Check Your Progress 2 :**

1 : c

**Check Your Progress 3 :**

1 : a

**Check Your Progress 4 :**

1 : d

**Check Your Progress 5 :**

1 : b

**Check Your Progress 6 :**

1 : d

#### **4.9 Glossary :**

1. **Package** – In Java, a package is a group of types that provides namespace management and access control.
2. **Interface** – An interface in programming is a collection of abstract methods that a class implements.

#### **4.10 Assignment :**

1. Explain object-oriented modeling and its benefits.

#### **4.11 Activities :**

1. Study the OOAD tools.

#### **4.12 Case Study :**

1. Discuss the difference between generalization and specialization and also explain how they are useful in object-oriented modeling.

#### **4.13 Further Readings :**

1. Learning Programming by Peter Norvig's.
2. Approach programming with a more positive by P. Brian. Mackey.



**UNIT STRUCTURE**

- 5.0 Learning Objective
- 5.1 Introduction
- 5.2 Importance of Modelling
- 5.3 Importance of Business Domain Modelling
- 5.4 Let Us Sum Up
- 5.5 Answers for Check Your Progress
- 5.6 Glossary
- 5.7 Assignment
- 5.8 Activities
- 5.9 Case Study
- 5.10 Further Readings

**5.0 Learning Objectives :**

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

- Modeling concept
- Understand the features of business domain modeling
- The Importance of Business Domain Modeling

**5.1 Introduction :**

There is a completely new way of thinking about problems with object-oriented modelling. With this approach, elements are visualised using models built around concepts from the real world. Object-oriented models help understand problems, communicate with remote experts, modelling companies, and design programmes and databases. In object-oriented modelling, objects and their properties are described. Any system has items that emerge to fill a role. Some object-oriented choices are used in the object role definition procedure.

The process of creating objects using object-oriented modelling (OOM) involves grouping together objects that have instance variables storing values. Models that are record-oriented contrast with object-oriented values, which only contain objects.

By combining application and database development, the object-oriented modelling method creates a single data model and language environment. In addition to allowing data abstraction, inheritance, and encapsulation, object-oriented modelling enables object identification and communication.

The process of planning and creating the actual structure of the model's code is known as object-oriented modelling. The object-oriented programming model is supported by the language used to implement the modelling techniques during the construction or programming phase.

Through three stages—analysis, design, and implementation—OOM develops object representation throughout time. Because the exterior components of the system are the main emphasis in the early stages of development, the model created is abstract. As the model develops, the primary emphasis moves from understanding how the system will be built to understanding how it should operate.

Object oriented modelling represents a completely new method of approaching issues. The main goal of this methodology is to visualise the data using models that are structured around notions from the real world. Object oriented models facilitate issue understanding, remote expert communication, modelling of businesses, and software and database creation.

We can all agree that creating a model for a software system before it is developed or transformed is just as important to its construction as having a major building's blueprint. Diagrams are used to represent object-oriented models. A good model is usually helpful for ensuring architectural soundness and for communication between project teams. It is crucial to remember that modelling techniques become more crucial as system complexity rises. Object Oriented Modeling is an appropriate modelling technique for managing a complicated system due to its properties.

OOM fundamentally entails creating a model of an application during system design that contains implementation specifics. As you are aware, the first three phases of the software life cycle—analysis, design, and implementation—refer to any system development. Instead of their final representation in any particular programming language, during object-oriented modelling applications are identified and organised according to their domain.

OOM is not language-specific, as may be seen. Any accessible programming language that is appropriate can be used to implement modelling once it has been completed for an application. The OOM method is one that encourages software developers to consider the application domain over the majority of the software engineering life cycle. The developer is compelled to recognise the fundamental ideas of the programme during this procedure. Before addressing the specifics of data structure and functionalities, the developer first organises and correctly understands the system.

## **5.2 The Importance of Modelling :**

Consider building a dog house, a home for your family, and an office building for a customer in order to understand the value of modelling. In the case of a dog house, little materials are required, and the dog's happiness is not very significant.

When designing a home for your family, you must take into account each member's needs and the available resources are not insignificant. The risk is extremely significant when developing a high rise office building.

Strangely, a lot of software development companies begin by wanting to construct skyscrapers but end up approaching the issue like they are pounding on a dog house. You can sometimes strike it lucky and have your team produce software that satisfies its users if the appropriate individuals are available at the right time and everything falls into place. This hardly ever occurs.

Successful software projects have many characteristics with unsuccessful programmes, while unsuccessful software projects all fail in different ways.

A great software organisation is made up of many different factors, but modelling is one of them.

A tried-and-true and widely used engineering technique is modelling. To assist their users in visualising the finished product, we create architectural models of homes and skyscrapers.

Modeling is not just used in the building sector. Aeronautics, automotive, visual arts, sociology, economics, software development, and many other fields use modelling. We create models so that we can test new theories or confirm existing ones with little risk and expense.

So what exactly is a model? Put simply,

**Real life is simplified in a model.**

A good model incorporates the major components and leaves out the minor components that are unimportant at the given degree of abstraction. A model could be behavioural, stressing the dynamics of the system, or structural, emphasising the way the system is organised.

We model because... We create models in order to better understand the system we are creating, and this is the only important motivation.

Four objectives are attained through modelling :

- With the aid of models, we may see a system as it is or as we want it to be.
- Models allow us to define a system's behaviour or structural characteristics.
- Models provide us with a blueprint that directs us as we build a system.
- Models serve as a record of our choices.

For one very simple reason, modelling becomes increasingly crucial as a system grows in size and complexity:

**We create models of complicated systems because we are unable to fully understand them.**

Modeling has advantages for all projects. By assisting them in creating the appropriate thing, modelling can speed up development by assisting the development team in better visualising the design of their system. The more complicated the project, the more likely it is that you will fail or, if you model at all, that you will create the incorrect thing.

- Graphical representations of the system to be constructed are provided via modelling.
- Modelling helps a software organisation succeed.
- Modelling is a tried-and-true engineering technique that is widely used.
- Modeling is not merely used in the construction sector. Without first creating models, from computer simulations to real wind tunnel models to full-scale prototypes, it would be impossible to launch a new aircraft or automobile. Real life is simplified in a model. The design of a system is provided by a model.
- A model may be structural, emphasizing the organization of the system, or it may be behavioural, emphasizing the dynamics of the system.

Models are created to assist us understand the system we are constructing better.

- a. They enable us to see the system as it is or as we would like it to be.
- b. Models allow us to describe the behaviour or structure of a system.
- c. Models provide us with a blueprint that directs us as we build a system.
- d. The models back up the choices we made.

Here are some of the benefits of the object-oriented approach:

➤ **Reduced maintenance :**

The first objective of object-oriented programming is to ensure that the system may have a longer lifespan while paying significantly less for maintenance. Because the majority of system operations are contained, behaviours are also reused and incorporated into new behaviours.

➤ **Real-world modeling :**

Compared to conventional approaches, object-oriented systems have a tendency to model the real world in a single, extremely comprehensive way. Classes of objects are used to categorise items, and behaviour is connected to objects. Instead than being based on data and processing, the model is built on objects.

➤ **Improved reliability and flexibility :**

Because new behaviours are "constructed" from existing objects, object-oriented systems promise to be significantly more dependable than conventional systems. New objects can also be produced at any time because the objects are dynamically called and accessed. Data properties from one or more other objects may be inherited by new objects. Additionally, behaviours are inherited from superclasses and new behaviours are added without impairing the operation of current systems.

➤ **High Code Reusability :**

Any new object that is formed immediately takes on the data attributes and properties of the class that it was derived from. Data and behaviour from any superclasses in which the new object participates will be inherited by it. A new object behaves "wigitty" despite having new behaviour defined in the system when a user creates a new widget type.

❑ **Check Your Progress – 1 :**

1. What are the benefits of object-oriented modeling ?
  - a. improved reliability
  - b. increased flexibility
  - c. reduced maintenance
  - d. all these
2. \_\_\_\_\_ is the process that groups data and procedures into an entity called objects.
  - a. Object development methodology
  - b. Linear programming
  - c. Structured programming
  - d. Object oriented system development

<b>5.3 Importance of Business Domain Modelling :</b>
--

We developed language as well as the ability to categorise the things in our environment as humans so that we could communicate ideas and concepts. Then we discovered how to write. and we still don't know when

## Object Oriented Analysis and Design

to stop! The quality and completeness of the requirements are frequently judged by how often the problem and need statements of our customers are refined. It almost seems that the quality of a document increases with its weight!

The goal of business modelling and our responsibility as business analysts is to provide a description of the company and its goals. A model is a simplification of a complex reality... so that we can better understand, according to The Unified Modelling Language (UML) User Guide. A model enables us to make sense of the business world and successfully explain concepts to others. It aids in the business owners' ability to visualise with us and to formally and precisely state their intentions. The firm finds it challenging to handle what they cannot see. Existing concepts and new requirements can be understood as being logically related to one another in a model.

Most business analysts are at least somewhat familiar with modelling the business process, which is where a company's beauty and distinctiveness are typically found. We are dealing with a dynamic perspective of the world—depicting the behaviour and activities of the business area of interest—if we can provide an answer to the query "what happens next?" The business activity requiring automated help, for instance, is identified in the following (simplified) illustration of an obsolete UK government business process related to the international commerce of food.

The external service bubbled the items that the underlying service opened, giving them the appearance of being outside. We can avoid the external client having excessive version control by integrating the domain model into the external service. Due to the external domain model being used by the external client, the internal service team is not required to maintain an older version of the service. In order to accomplish particular goals that are carried out under the owner's responsibility, the business domain engages in business processes, also referred to as the transition of something from one state to another via coordination.

Good information system software will be utilised to support the business domain. This software will assist the firm in working through its internal business process and controlling all the factors that influence the process. Good business process modelling and implementation methodologies that expertly assess, model, and implement business processes in order to meet organisational goals should be used to assist the business process. There are many businesses where we find fields like logistics and processing that deal with product knowledge in addition to economics.

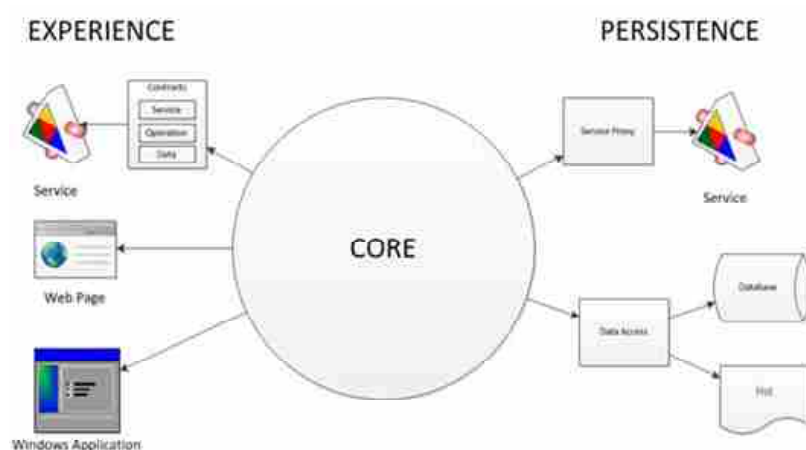


Figure 5.1 : Business domain model

Every domain has particular requirements, such as: Orders : Demand complete customer information even while not all details are required, which aids in representing the client in many domain models. Warehouse: Here, only the product specifics and logistics information are required from the buyer in order to collect the products. Shipping: The buyer supplies name, address, and contact information in the event of shipping.

The application, persistence, and experience in the business domain model depicted in Figure 5.1 are examples of extensions to the application itself that are used to accomplish the desired results.

Here, the data layer is renamed to persistence, which is Core as an application model, with the justification that a persistent data-based application will not be subject to change. When moving from a persistent file-based strategy to a database, just your access technique needs to change.

The Core as Application model is the new term for the business level. This is so because the application is contained in this level. No changes are made to the app or persistence strategy when the experience layer is altered.

❑ **Check Your Progress – 2 :**

1. The business process is related to:
  - a. structured set of activities designed to produce the necessary output for the market
  - b. structured set of activities designed to produce the result that the customer requires
  - c. Both a and b
  - d. neither a nor b
2. Point out the Correct statement
  - a. The UML standard is the work of the Object Management Group
  - b. SysML creates graphical representations of software systems in the form of a set of diagram types
  - c. UML diagrams are separated into four structural types and seven behavior types
  - d. All of the mentioned
3. Which of the following combines a modeling language with a graphical display of the various SOA components ?
  - a. Business Process Modeling Notation
  - b. Service-Oriented Modeling Framework
  - c. Systems Modeling Language
  - d. All of the mentioned

➤ **Domain driven modeling :**

Having a business process defined as the change of anything from one state to another by partially coordinated agents, with the aim of attaining goals derived from process owner accountability, the business domain of any organisation fits into the organization's business process. Business processes are described as "organised sets of operations designed to deliver a certain result for a particular consumer or market," for instance.

Different business domains operating in the same business industry share comparable business procedures. Good information system software supports the business domain by managing internal business processes and controlling all factors that have an impact on how those processes are carried out.

#### **5.4 Let Us Sum Up :**

While studying this device, we learned that object-oriented modeling is a whole new way of thinking about problems. This method is about visualizing elements through the use of models organized around ideas from the real world. The business domain is seen to carry out business processes where it transforms something from one state to another using coordination to have ownership responsibility goals.

#### **5.5 Answers for Check Your Progress :**

**Check Your Progress 1 :**

1 : d            2 : d

**Check Your Progress 2 :**

1 : c            2 : a            3 : b)

#### **5.6 Glossary :**

1. **Design :** in modeling, problem related to the domain of objects in programming.
2. **Domain Model :** a tool helps to frame the conceptual understanding of product el system.
3. **Object Design :** It is a design strategy that system designers think about. operations or features.

#### **5.7 Assignment :**

1. Write a short note on Enterprise Domain Modeling.

#### **5.8 Activities :**

1. Collect domain modeling information.

#### **5.9 Case Study :**

1. Generalize the basic architecture of the computer and discuss.

#### **5.10 Further Readings :**

1. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1995.
2. O.M. Nierstrasz, A Survey of Object-Oriented Concepts. In W. Kim, F. Lochovsky (eds.): Object-Oriented Concepts, Databases and Applications, ACM Press and Addison-Wesley, 1989.
3. Wade, S., Salahat, M., Wilson, A Scaffolded Approach to Teaching Information Systems Design, 2012
4. Salahat, M., Wade S. (2014) Teaching Information Systems Development Through an Integrated Framework, Oxford University, Oxford, UK.
5. Williams B. (2005) Soft Systems Methodology, 2010.

**UNIT STRUCTURE**

- 6.0 Learning Objective
- 6.1 Introduction
- 6.2 Aggregation
- 6.3 Abstract Class
- 6.4 Multiple Inheritance
- 6.5 Generalization as an Extension
- 6.6 Generalization as a Restriction
- 6.7 Metadata
- 6.8 Constraints
- 6.9 Let Us Sum Up
- 6.10 Answers for Check Your Progress
- 6.11 Glossary
- 6.12 Assignment
- 6.13 Activities
- 6.14 Case Study
- 6.15 Further Readings

**6.0 Learning Objectives**

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

- Study on the characteristics of the object model.
- Learn more about the advantages of the Abstract Class
- Multiple Inheritance Study
- Extension and restriction study

**6.1 Introduction :**

The process through which one thing gains the qualities of another is frequently described as inheritance. Information is made manageable in a hierarchical structure through the use of inheritance. The least frequently used keywords in relation to inheritance would be extend and implement. These phrases would establish whether one object is in the same style as another. We will design an object to take the properties of another object using these terms. A mechanism used at build time may be inheritance. A superclass will have any number of subclasses. A subclass, however, will only have one superclass. Java does not enable multiple inheritances, which is why.

**6.2 Aggregation :**

The notion that aggregation is a powerful form of association in which components are put together to create an aggregate object. Aggregation is an



extension of association. The components are a part of the aggregate. The transition between "part" and "whole" or "apart" is made via aggregation. Aggregates, a special case of association, are a directional association between things. When one thing "has-a" the other, there is an aggregate between the two. The orientation between them makes it obvious that one thing is contained by the other. Aggregation is also known as a "Has-a" relationship.

❑ **Check Your Progress – 1 :**

1. Aggregation is a \_\_\_\_\_
  - a. Is a relationship
  - b. a part of the relationship
  - c. has a relationship
  - d. b and c both

**6.3 Abstract Class :**

In Java, a class is a group of objects that have certain common properties. It is a kind of model or model from which objects are created. So a class in java contains :

- Data member
- Method
- Constructor
- Block
- Class and interface

The syntax for declaring a class is:

```
Class < class_name > {  
data member;  
Method  
}
```

➤ **Consider an example of Object and Class :**

This example creates a Student class with the ID and name which are two data members. Now the Student class object is created with the help of a new keyword and prints the object values as:

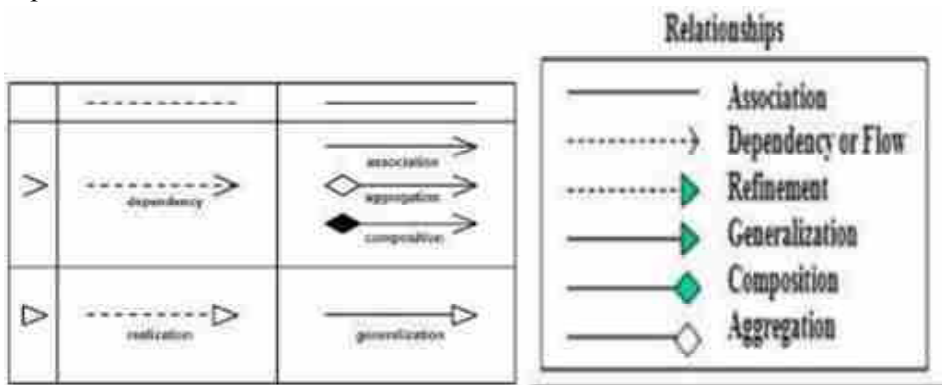
```
Class Student {  
int id;  
String name;  
public static void main ( String args []) {  
Student s = new Student ( );  
System.out.println( s1.id );  
System.out.println ( s1.name);  
}  
}
```

If we run the above program, we find:

Output: 0 null

➤ **Abstract Class :**

An abstract concept that cannot be instantiated is defined by an abstract class. Abstract class objects can't be made; they will only inherit. Typically, an abstract class describes a concept that has a set of general actions attached to it. As long as the interfaces are implemented, the abstract class can only be inherited. It cannot be created from scratch. The interfaces of the abstract class's implemented methods will only contain method declarations without implementations.



**Figure 6.1 : Abstract class relationship**

❑ **Check Your Progress – 2 :**

1. Abstract classes are \_\_\_\_\_
  - a. Inherited
  - b. Implemented
  - c. Instantiated
  - d. None of these

**6.4 Multiple Inheritances :**

The most crucial aspect of object-oriented programming is inheritance, which enables one class or category to use the properties and functions of another type. In this, the base class is referred to as a superclass, while the derived class is referred to as a subclass. It can be observed that a derived class includes extra variables and methods that set it apart from the base class. The syntax is demonstrated below:

```
Public class ChildClass extends BaseClass {
    extended and possibly overridden derived class methods
}
```

Consider an example :

```
class Vehicle {
    String color;
    int speed;
    int size;
    void attributes() {
        System.out.println("Color :" + color);
        System.out.println("Speed :" + speed);
        System.out.println("Size :" + size);
    }
}
```

**Object Oriented  
Analysis and Design**

```
// A subclass which extends for vehicle
class Car extends Vehicle {
    in CC;
    int gears;
    void attributescar() {
        // The subclass refers to the members of the superclass
        System.out.println("Color of Car :" + color);
        System.out.println("Speed of Car :" + speed);
        System.out.println("Size of Car :" + size);
        System.out.println("CC of Car :" + CC);
        System.out.println("No of gears of Car :" + gears);
    }
}

public class Test {
    public static void main{String args[]} {
        Car b1 = new Car();
        b1.color = "Blue";
        b1.speed = 200 ;
        b1.CC = 1000;
        b1.gears = 5;
        b1.attributescar();
    }
}
```

If we run the above program, we get :

Car color : blue.

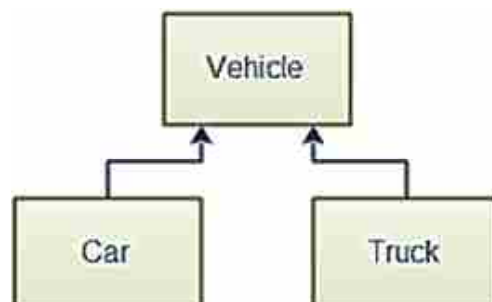
Car speed : 200

Car size : 22

Car CC : 1000

Number of gears of the car : 5

It turns out that inheritance is a useful technique for sharing code across several classes that share some qualities, allowing the classes to have separate portions. A vehicle class, represented in Figure 6.2, has two subclasses, such as Cars and Trucks.



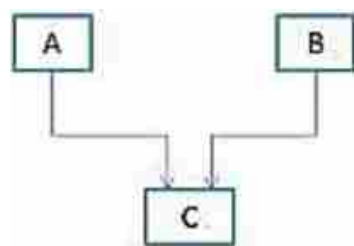
**Figure 6.2 : Vehicle class**

As can be seen, the vehicle class is a superclass of the vehicle subclasses cars and trucks. Here, the Car and Truck classes have particular fields and methods for cars and trucks, while the Vehicle class provides the necessary fields and methods for vehicles.

It has been observed that many people argue that one way to categorise a group of people is by inheritance. According to studies, a truck and an automobile are both types of vehicles. So you cannot choose superclasses and subclasses in your application in this manner. Just demonstrate how you must cooperate with them.

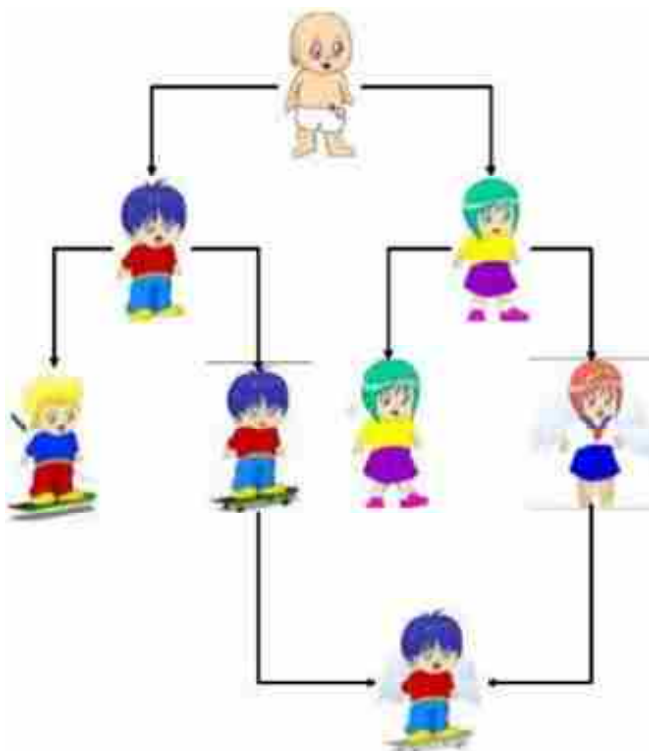
As you can see, all public and protected fields and methods are inherited from a subclass when it extends a superclass. As if the subclass were declaring itself, the fields and methods are included in this.

Using the multiple inheritance technique, a class can extend from multiple base classes.



**Figure 6.3 : Multiple inheritance**

The final point, which is referred to as multiple inheritance, is an advanced kind of inheritance. An alternative strategy is provided by multiple inheritance, where the new class can take information from both pertinent classes. Figure 6.4 illustrates this.



**Figure 6.4 : Multiple inheritance**

The sloopy below has wheels and wings that it inherited from the rim sloopy and the float sloopy, which is inconsistent with the depiction above.

In this, the boy, not the girl, is the sloopy, and the wheels of the sloopy also have characteristics acquired from the boy and girl, who seem to be at odds.

**❑ Check Your Progress – 3 :**

1. If class A is inherited from class B, choose the correct option from the following.
  - a. Class B + Class A
  - b. Class B inherits class A
  - c. Class B extends A
  - d. Class B extends class A

**6.5 Generalization as an Extension :**

The process of creating a superclass from a set of items with similar semantic properties is known as generalisation. Extracting properties that are shared by two or more classes and integrating them into a generalised superclass is the process of generalisation. Attributes, affiliations, or procedures can all be considered shared qualities.

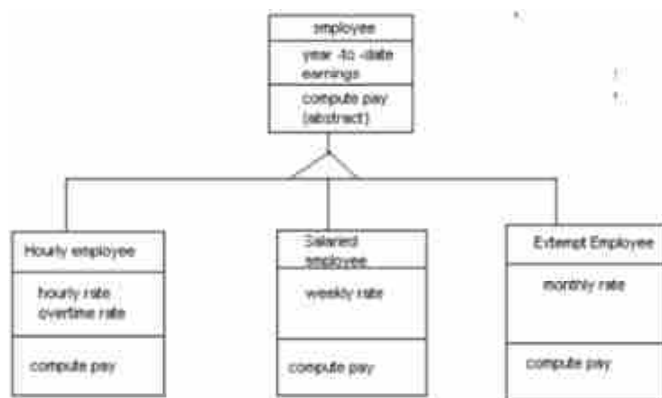
The relationship between a class and one or more improved variants of it is known as generalisation. Each improved version is referred to as a "subclass," while the refined class is referred to as a "superclass." Equipment, for instance, is the super class of a bomb and a tank. Each subclass shares the superclass's associated attributes and functions, which are shared by a group of related subclasses. The "is-a" relationship is another name for the generalisation. A subclass's instances are copies of its instances of the superclass.

**❑ Check Your Progress – 4 :**

1. Generalization is a \_\_\_\_\_
  - a. Relation between a class and its refined versions
  - b. is a relationship
  - c. Defining super class from given set of related entities
  - d. All of these

**6.6 Generalization as a Restriction :**

In general, an instance of a class consists of all of its predecessors as well as its current instance. Therefore, you could claim that all of the parent class's features must be used by subclass instances. This includes both operations on the ancestor class and the ancestor class's attributes. Extensions are added features that a subclass can have. For instance, Figure 6.5 adds three subclasses to the Employee class that both inherit all of Employee's functions and create new ones of their own.



**Figure 6.5 : Employee class**

The qualities of ancestors may also be restricted by a subclass. Because it restricts the possible values for instances, this is known as a constraint. For instance, an ellipse with equal major and minor axes is a circle. A constrained subclass's constraints may be broken by arbitrary modifications to its attribute values, in which case the outcome will no longer be a member of the original subclass. Since the outcome is still a legitimate superclass instance, this is not a problem from the perspective of the superclass.

**❑ Check Your Progress – 5 :**

1. What is meant by restriction on generalization ?
  - a. Attribute
  - b. All the functionality of the parent class must be applied to the instances of the subclass, this is known as a restriction.
  - c. Inherited features
  - d. None of those

**6.7 Metadata :**

Data that describes other data is referred to as metadata. Since metadata is also data, it goes without saying that a lot of data is useless without it. Basic information about the data is summarised in the metadata, which might make it simpler to locate and use specific instances of the data. Metadata is also utilised for photos, movies, spreadsheets, and web pages in addition to document files. Metadata usage on web pages has a lot of potential.

The metadata for a web page includes both descriptions of the page's content and keywords that are pertinent to it. Typically, meta tags are used to express them. The correctness and detail of the metadata, which includes the description and summary of the web page, are particularly essential since it might affect a user's decision to visit the web page or not. Metadata are frequently presented in the search results of search engines.

**❑ Check Your Progress – 6 :**

1. Which of the following statements is true for metadata ?
  - a. It is the data that describe other data.
  - b. Basic information summarized on the data.
  - c. It is expressed in the form of a meta tag.
  - d. All of these

**6.8 Constraints :**

A restriction could be a relationship between items that is numerical or geometric. Declarative language describes constraints. A natural method to describe interactions between items is through constraints. It seems challenging to combine constraint systems and object-oriented programming (OOP). The OOP encapsulation principle is an implicit commitment made by all current systems. Without specifying how they should be recorded, constraints express properties that must always be true for the entire system. Most often, informal text, operational restrictions, or the incorporation of constraints into preexisting model concepts are used to specify constraints. With constraints, you can define the system's general characteristics without defining how they will be implemented.

**Check Your Progress – 7 :**

1. What can constitute constraints ?
  - a. numeric data
  - b. geometric relationship
  - c. Both of these
  - d. None of these

**6.9 Let Us Sum Up :**

In this unity we learned that inheritance can be defined as the process in which one object acquires the properties of another. These inheritances ensure that information is managed in a hierarchical order. It has been studied that a subclass in Java is an inheritance method that derives from the Java superclass. There are three types of variables in Java such as Local, Instance and Static. It is studied that the local variable is a variable that is declared inside the method itself, the instance variable is a variable that is declared inside the class but outside the method and the static variable is a variable that is declared static is called static variable. In Java, it is seen that the constructor can be inherited since the constructor name is based on the class name. When inheriting methods, the method signature must remain the same.

**6.10 Answers to Check Your Progress :**

**Check Your Progress 1 :**

1 : d

**Check Your Progress 2 :**

1 : a

**Check Your Progress 3 :**

1 : c

**Check Your Progress 4 :**

1 : d

**Check Your Progress 5 :**

1 : b

**Check Your Progress 6 :**

1 : d

**Check Your Progress 7 :**

1 : c

**6.11 Glossary :**

1. Static variable : it is a variable declared static and called a static variable.
2. Class: are a group of objects with common properties and are like a model from which objects are created.

**6.12 Assignment :**

1. Write a short note about inheritance and its use.

**6.13 Activities :**

1. Write a short note about inheritance and its use.

**6.14 Case Study :**

1. Discuss the result of this program

```
class A {
    int i;
    int j;
    A() {
        i = 1;
        j = 2;
    }
}
class Output {
    public static void main(String args[])
    {
        A obj1 = new A();
        A obj2 = new A();
        System.out.print(obj1.equals(obj2));
    }
}
```

**6.15 Further Readings :**

1. DOWLATSHAHI, S., 1992, Product design in a concurrent engineering environment: an optimization approach, International Journal of Production Research, 30(8), pp. 1803–1818.
2. ESCHENAUER, H., KOSKI, J. and OSY CZKA, A., 1990, Multicriteria Design Optimization: Procedures and Applications (New York: SpringerVerlag).
3. FENG, C. X. and KUSIAK, A., 1995, Constraint-based design of parts, Computer-Aided Design, 27(5), pp. 343–352.



**BLOCK SUMMARY :**

In this block, you learned about inheritance and its relationship to the subclass and interface block in detail. Variable type concepts with illustrations are explained step by step. The block emphasizes package and interface information with examples of inheriting abstract interface methods. Information about declaring an interface reference variable is well explained with examples. You also learned the importance of modelling.

After studying this block, you will feel confident while working on a simple Java platform and you will be able to improve your knowledge by studying some examples and illustrations mentioned in this block. With such detailed knowledge of inheritance, interface, packages, and exceptions in Java, you can take advantage of it in the future.

<b>BLOCK ASSIGNMENT :</b>
---------------------------

❖ **Short Questions :**

1. What is Object Oriented Modeling ?
2. Explain inheritance and its types.
3. What is the difference between generalization and specialization ?
4. Write a short note about abstract classes.

❖ **Long Questions :**

1. What is multiple inheritance ? Explain in detail with the help of an example.
2. Write a short note about the metadata.
3. Write a note on generalization as a restriction.

**Object Oriented  
Analysis and Design**

❖ **Enrolment No. :**

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

Unit No.	4	5	6
No. of Hrs.			

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

Items	Excellent	Very Good	Good	Poor	Give specific example if any
Presentation Quality	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Language and Style	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Illustration used (Diagram, tables etc)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Conceptual Clarity	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Check your progress Quest	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Feed back to CYP Question	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____

3. Any other Comments

.....

.....

.....

.....

.....

.....

.....

.....



**Dr. Babasaheb Ambedkar  
Open University Ahmedabad**

**BCAR-502**

# **OBJECT ORIENTED ANALYSIS AND DESIGN**

---

## **BLOCK 3 : UNIFIED MODELLING LANGUAGE-I**

---

UNIT 7 INTRODUCTION TO UML

UNIT 8 BASIC AND ADVANCED STRUCTURE MODELLING

UNIT 9 CLASS & OBJECT DIAGRAM

UNIT 10 BASIC BEHAVIOURAL MODELLING-I

# ***UNIFIED MODELLING LANGUAGE–I***

## **Block Introduction :**

An overall, graphical modelling language used in software engineering is called Unified Modelling Language (UML). The software system's artefacts are specified, visualized, built, and documented using UML. Grady Booch, Ivar Jacobson, and James Rumbaugh created it at Rational Software in 1994–1995 and continued to work on it until 1996. It was made a standard by the Object Management Group in 1997.

In this block we will detail the importance of modelling, the principle of modelling, object-oriented modelling, UML conceptual model and its architecture. We will also focus on common classes, relateder, mechanisms and diagrams, benefits of classes, Advanced relationship used in different diagrams, interfaces, role types and packages. We will also focus on class and object diagrams in detail, such as their terms, concepts, different modelling techniques for class and object diagrams. We will also discuss the details of the basic concepts of behaviour modelling. In it we will see the details of the interaction and different interaction diagrams.

## **Block Objectives :**

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

- About importance of modelling,
- About modelling principles,
- About object-oriented modelling,
- About the conceptual model of the UML
- About Classes, Relationship, Mechanisms and Diagrams
- About class and object diagrams
- About behaviour modelling

**Block Structure :**

**Unit 7 : Introduction to UML**

**Unit 8 : Basic and Advanced Structure Modelling**

**Unit 9 : Class & Object Diagram**

**Unit 10 : Basic Behavioural Modelling–I**

**UNIT STRUCTURE**

- 7.0 Learning Objectives
- 7.1 Introduction
- 7.2 The Importance of Modelling
- 7.3 Principles of Modelling
- 7.4 Object Oriented Modelling
- 7.5 Conceptual Model of the UML
  - 7.5.1 Building Blocks of UML (Conceptual Model of UML)
    - 7.5.1.1 Things
    - 7.5.1.2 Relationships
    - 7.5.1.3 Diagram
  - 7.5.2 Rules
  - 7.5.3 Common Mechanisms in UML
- 7.6 Architecture
- 7.7 Let Us Sum Up
- 7.8 Answers to Check Your Progress
- 7.9 Glossary
- 7.10 Assignment
- 7.11 Activity
- 7.12 Case Study
- 7.13 Further Readings

**7.0 Learning Objectives**

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

- Idea of Importance of modelling
- Detail of Major of modelling
- Object Oriented Modelling
- Conceptual model of the UML
- Architecture

**7.1 Introduction :**

UML stands for Unified Modelling Language, a common visual modelling language standardized in the engineering field. It is used to specify, display, build and document the most important artifacts in the software system. It helps to design and characterize, especially the software systems that incorporate the concept of object orientation. Describes derives from a system of software and hardware.



In the reasons why, in UML has become a standard model Languages that it is independent of the programming language. Also, the UML notation set is a language and not a method. This is important because a language, unlike a methodology, can easily fit into any business's way of doing business without the need for change.

Since UML is not a method, it does not require any formal work product. However, it provides different types of diagrams which, when used in in method data, increase the understanding of in application under development. There is more to UML than these diagrams, but for my purpose here, diagrams provide a good introduction to the language and the principles behind its use. By inserting standard UML charts into your methodological work products, you make it easy for people with UML skills to participate in your project and quickly become productive. The most useful standard UML diagrams are: Utilize class, sequence, state, activity, component, and implementation diagrams, as well as case diagrams.

## **7.2 The Importance of Modelling :**

To learn the importance of Modelling, let's say you need to build a dog house, a house for your family, and a high-rise office for a customer. In the case of a dog house, you need minimal resources and dog satisfaction is not that important.

In the case of building a house for your family, you need to meet the requirements of your family members and the amount of resources is not trivial. In the case of building a high-rise office, the risk is very high.

It's interesting how many software development companies begin by wanting to construct skyscrapers but end up approaching the issue as they're knocking on a dog's door. You can be cruel at times. You might be able to persuade your team to produce a software product that is well-liked by its users if the correct people are present at the right moment, and if the two planets are in the right positions. This hardly ever occurs. Software initiatives that fail do so in their own particular way, whilst those that succeed do so in a variety of ways. A great software organisation is made up of a variety of factors, but modelling is one of them.

Modelling is a well-accepted and proven engineering technique. We build architectural models of houses and skyscrapers to help your users visualize the final product. Modelling is not just limited to the construction industry. Modelling is used in the areas of aviation, automotive, photography, sociology, economics, software development and many more. We create models so that we can test new hypotheses or confirm existing ones with little risk and expense.

Complete modelling, we accomplish four goals :

- Models enable us to see a system in its current or desired state.
- We can specify a system's structure or behaviour using models.
- Models give us a blueprint to use while we construct a system.
- Models serve as a record of our choices.

Modelling can be useful for any job. By assisting them in creating the proper thing, modelling can help the development team better visualise their system design and enable them to evolve morally. If you model a project, the more complicated it is, the more likely you are to fail or create anything incorrectly.

❑ **Check Your Progress – 1 :**

1. Which model from the following depicts the dynamic behaviour in system modelling ?
  - a. Behavioural Mode
  - b. Data Model
  - c. Context Model
  - d. Object Model

**7.3 Principles of Modelling :**

All engineering fields have a long history with modelling. These observations indicate four fundamental modelling ideas.

• **First Modelling Principle :**

**"How an issue is approached and how a solution is shaped has a major impact on which models to generate"**

Make wise model selections. The worst development issues will be brought to light by the appropriate models. You'll be led astray by the incorrect models into concentrating on pointless subjects.

• **Second modelling principle :**

**"There are various levels of precision at which each model can be expressed."**

Sometimes all you need is a user interface executable model that is quick and simple. At other times, you must dig into intricate details, such as system interfaces or network problems, etc.

In any case, the requested types of models are the ones that allow you to choose your level of detail, depending on who is looking at it. An analyst or end user would like to focus per what questions, or in developer would like to focus per how questions.

• **Third Modelling Principle :**

**"The requested models are connected to reality."**

Software should have a closer relationship between the analysis model and the system design model. If this gap cannot be filled, the system eventually diverges. All the nearly independent perspectives of a system can be connected to a whole in object-oriented systems.

• **Fourth principle of modelling :**

**"No model is sufficient. The ideal way to approach any non-trivial system is with a condensed set of almost independent models."**

You can examine the electrical plans for a building alone, but you can also see how they relate to the floor plan and potentially even how they interact with the plumbing plan's piping.

❑ **Check Your Progress – 2 :**

1. "Each model can be expressed at different levels of precision" which principal is it ?
  - a. First
  - b. Second
  - c. Third
  - d. Forth

**7.4 Object Oriented Modelling :**

Links and association are ways of creating links that exist between objects and classes. In software, there are several ways to approach the construction

## Object Oriented Analysis and Design

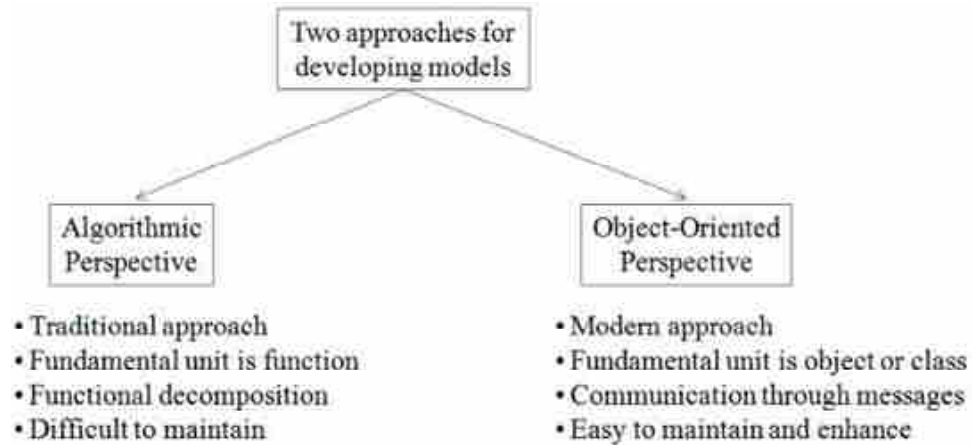
of a model. Since a most common ex is : from an **algorithmic perspective and from an object-oriented standpoint.**

An algorithmic viewpoint characterises the conventional understanding of software development. According to this method, the technique or function of any software serves as its primary component. Developers are prompted by this viewpoint to concentrate on control problems and the breakdown of complex algorithms into simpler ones.

Systems created utilising an algorithmic approach become particularly challenging to maintain when the needs change and the system expands.

The (other) modern viewpoint on software development is object-oriented. The object or class is the central element of tutti software systems in object-oriented modelling.

In a nutshell, an object is a thing that is usually drawn from the elements of the problem space or solution space. A class is a description of a set of common objects. Each object has state, identity, and behaviour.



**Figure 7.1 : Approaches to model development**

### ❑ Check Your Progress – 3 :

1. UML diagram includes \_\_\_\_\_
  - a. Class name
  - b. List of operations
  - c. (a) and (b) both
  - d. None of these

### **7.5 Conceptual Model of the UML :**

To learn a new language, consider for example English, we first learn the character set (az) and then the words and then different rules for forming sentences using these words.

Similarly, to learn how to use UML, we need to learn the vocabulary and then the rules for making UML diagrams. The conceptual model for UML contains the vocabulary and rules of UML. So, to understand the use of UML and create using UML diagrams, we must first learn the UML conceptual model.

The conceptual model of UML contains the basics of UML. The conceptual model consists of three parts :

- (1) Building blocks of UML (syntax / vocabulary)
- (2) Rules (semantics)
- (3) Common Mechanisms

### 7.5.1 Building Blocks of UML (Conceptual Model of UML) :

These are the fundamental elements in UML. Every diagram can be represented using these building blocks. The building blocks of UML contains three types of elements. They are :

- (1) Things (object-oriented parts of UML)
- (2) Relationships (relational parts of UML)
- (3) Diagrams

#### 7.5.1.1 Things :

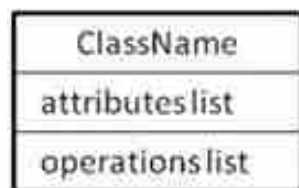
A chart can be seen as a graph that contains points and edges. In UML, corners are replaced by things and edges are replaced a relational. There are four things in UML. Da um :

- (a) Structural things (UML nouns – static parts)
- (b) Behaviour (UML verbs – dynamic parts)
- (c) Groups of things (organizational parts)
- (d) annotation (explanatory part)

#### (a) Structural thing :

It represents the static aspects of a software system. There are seven structural elements in UML. Da um:

**Class** – A class is a collection of similar objects that have similar properties, behaviours, parentheses, and semantics. Graphically, the class is represented as a rectangle with three spaces as shown in Figure 7.2



**Figure 7.2 : Graphic representation of the class**

**Interface** – An interface is a collection of function signatures and / or attribute definitions that ideally define a coherent set of behaviours. Graphically, the interface is represented as a circle or a stereotypical class symbol with interface as shown in Figure 7.3.



**Figure 7.3 : Graphic representation of the interface**

**Use Case** – Use case is a collection of actions that define the interactions between a role (actor) and the system. Graphically, the use case is represented as a solid ellipse with the name written inside or below the ellipse.



**Figure 7.4 : Graphic representation of the use case**

**Collaboration** – A collaboration is the collection of interactions between objects to achieve a goal. Graphically, the collaboration is represented as a dotted ellipse. A collaboration can be a collection of clusters or other elements.



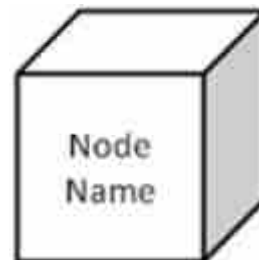
**Figure 7.5 : Graphic representation of collaboration**

**Component** – A component is a physical and useable part of a system. Graphically, the component is represented as a tab-shaped rectangle. Examples per component executable files, DLL files, database tables, files e documents.



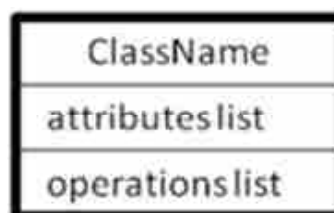
**Figure 7.6 : Graphic representation of components**

**Node** – A node is a physically present sub-driving component that stands in for a computational resource. Graphically, the node is represented as a cube. Examples of nodes are PCs, laptops, smartphones or any embedded system.



**Figure 7.7 : Graphic representation of the node**

**Active Class** – a class whose objects can start their own flow of control (threads) and work in parallel with other objects. The graphically active class is represented as a rectangle with thick edges.



**Figure 7.8 : Graphic representation of the active class**

**(b) Behavioural things :**

It represents the dynamic aspects of a software system. The behaviour of a software system can be modelled as interactions or as a sequence of state changes.

➤ **Interaction :**

A behaviour made up of a series of messages exchanged between a number of objects in order to carry out a particular task bent into the shape of a continuous arrow in morse. Here is an instance of an interaction that simulates a phone call :

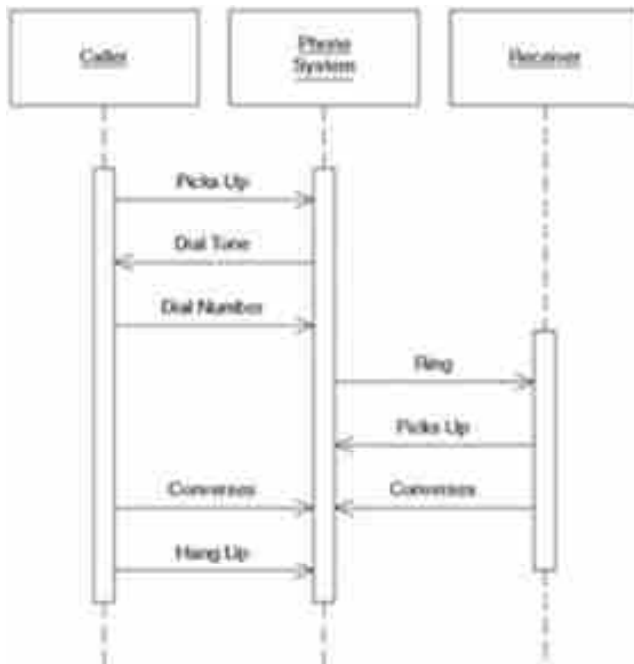


Figure 7.9 : Interaction

**State Machine** – In behaviour that outlines the series of conditions that an object or interaction goes through throughout the course of its lifetime in response to circumstances. In state is represented as a rectangle with rounded corners. Below is an example of a state machine representing the states of a telephone system:

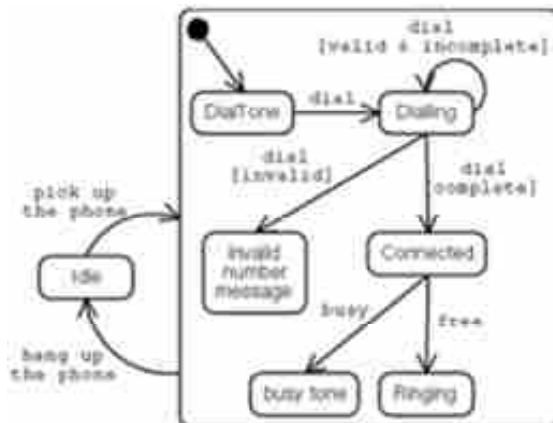


Figure 7.10 : State machine

(c) **Grouping things :**

Elements that are used fine in the organization of relationships and relationships with modelers.

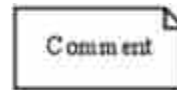
**Package :** In general mechanism for organizing goods into groups. Graphically, the package is represented as a folder with tabs. As charts become large and cluttered, the related ones are grouped into one package, making the chart less complex and easier to understand.



Figure 7.11 : Graphic representation of packages

**(d) A notational thing :**

A symbol for displaying a comment it is used. Graphically, the note is represented as a rectangle with a dog ear in the upper right corner.



**Figure 7.12 : graphical note**

**7.5.1.2 Relationships :**

Things in a diagram are connected through relational. So, one relates a connection between two or more things.

**Dependency :** a semantic connection where a change in one item affects the other (the independent thing) can cause changes in the other thing (the dependent thing). This relationship is also known as the "usage ratio". Graphically shown as a dotted line with an arrowhead as shown in Fig. 7.13.



**Figure 7.13 : Dependency representation**

**Association :** A structural relationship that describes connections between two or more things. Graphically represented as a solid line with an optional arrow representing navigation.



**Figure 7.14 : Example of Association**

**Generalization :** It is a Generalization–specialization relationship. In a nutshell, this describes the relationship between a main class (generalization) and its subclasses (specializations). Also known as the "is–one" relationship.



**Figure 7.15 : Graphic representation and generalization**

**Realization :** Defines a semantic relationship in which one class specifies something to be performed by another class. Example: The relationship between an interface and the class that implements or executes that interface.



**Figure 7.16 : Graphical representation of the realization**

**7.5.1.3 Diagram :**

A diagram is a collection of elements that are often represented as a graph consisting of corners and edges that connect these corners. These corners in UML are things, and the edges are relations.

1. Use Case Diagram
2. Sequence Diagram

3. Activity Diagram
4. Class Diagram
5. Collaboration Diagram
6. State Diagram
7. Object Diagram
8. Component Diagram
9. Implementation Diagram

### 1.5.2 Rules :

The rules for UML specify how the building blocks of UML are put together to develop diagrams. Rules allow users possible per create well-formed models. In well-shaped model is self-consistent and also consistent with the other models.

UML will rule for :

Names: what elements can be called things, contexts e figures

Scope: The context in which a name has a particular meaning.

Visibility: how these names are seen and how they can be used by other names

Integrity: It is the correct relationship between things.

Execution: what it means to run or pretend a model

### 1.5.3 Common Mechanisms in UML :

The four common mechanisms used in UML are:

specification

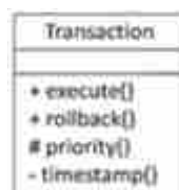
Adornments

common divisions

expansion mechanisms

**Specifications** : Behind every graphic notation in UML is a precise specification of the hoists that the element represents. For example, a class icon is a rectangle that specifies the class name, attributes, and operator.

**Adornments** : The mechanism of the UML that allows users to specify additional information with the basic notation of an element is embellishment.



**Figure 7.17 : Ornament example**

In the example for no per, the access specifications represent: + (public), # (protected) and – (private) the visibility of the attributes, which is further information on top of the basic representation of the attribute.

**Common divisions** : In UML, there is a clear division between semantically relational elements such as: the separation between a class the separation between an interface and its implementation.





**Figure 7.18 : Example of common division**

**Extensibility Mechanisms**

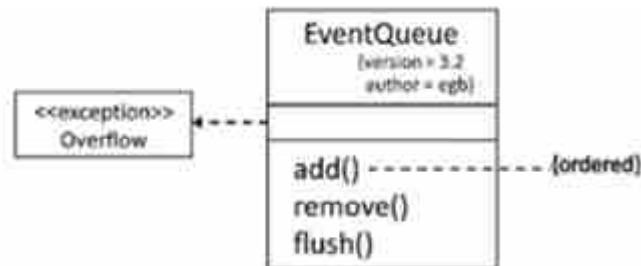
The UML extension mechanisms allow the user to expand (new additions) the language in a controlled way. The extension mechanisms in UML are:

**Stereotypes** – expands vocabulary per UML. It allows users to declare new building blocks (icons) or expand the basic notations of existing building blocks by stereotyping them using guillemets.

**Tagged values** – Expands the properties per in UML building block. It allows us to enter additional information in the item specification. Represented as text enclosed in parentheses and placed the sotto element name. Properties are :

{property name = value}

**Constraints** – Extends the semantics of a basic UML component, such as specifying new rules or modifying existing rules. Represented as text enclosed in curly braces and placed next to or next to the item name.



**Figure 7.19 : Expansion mechanism**

In the example above, we specify the "Overflow" exception using the class symbol and write it in stereo with "exception". Also, the sotto class name " Event Queue " we specify additional properties like "version" and "author" using tagged values.

**❑ Check Your Progress – 4 :**

1. \_\_\_\_\_ from the following applies to a class rather than object.
  - a. Constructor
  - b. Update
  - c. query
  - d. scope

**7.6 Architecture :**

A model is a simplified representation of the system. To visualize a system, we will build various models. The subgroup of these models is a view. Architecture is a collection of different views.

The many perspectives will be of interest to the company's stakeholders (end users, analysers, programmers, technology providers, testers, technical writers, and project managers).

The architecture can best be represented as a collection of five views : (1) Use case view, (2) Promethazine / logic view, (3) Implementation / Development view, (4) Process view, and (5) Implementation / Physical view.

Since five views can be summarized as shown in the following table :

**Table 7.1 of 5 views of the architecture**

View	Stake holder	Static Aspects	Dynamic Aspects
Use case view	End users Analysts	Use case diagrams	Interaction diagrams State chart diagrams Activity diagrams
Design view	End users	Class diagrams	Interaction diagrams State chart diagrams Activity diagrams
Implementation view	Programmers	Component diagrams	Interaction diagrams State chart diagrams Activity diagrams
Deployment view	System Engineer	Deployment diagrams	Interaction diagrams State chart diagrams Activity diagrams

**❑ Check Your Progress – 5 :**

- In UML diagram, \_\_\_\_\_ used to show the relationship between object and component parts.
  - aggregation
  - increment
  - ordination
  - segregation
- \_\_\_\_\_ diagram show interaction between messages.
  - activity
  - collaboration
  - state chart
  - object lifeline

**7.7 Let Us Sum Up :**

In this unit, we have learned what the unified modelling language is and what is the importance of modelling for application development. We also learn four main modelling with explanation. We have also discussed object-oriented modelling with two different approaches to developing any model.

We have also discussed the conceptual model of UML explanation of its element as things (object-oriented parts of UML), reporter e diagrams used in UML. Finally, we have also discussed the architecture of UML.

**7.8 Answers for Check Your Progress :**

- ❑ Check Your Progress 1 :**  
1 : a
- ❑ Check Your Progress 2 :**  
1 : b
- ❑ Check Your Progress 3 :**  
1 : c
- ❑ Check Your Progress 4 :**  
1 : d
- ❑ Check Your Progress 5 :**  
1 : a      2 : b

**7.9 Glossary :**

1. **Package** – It is a collection of types that gives access protection and name space management in Java.
2. **Architecture** – Architecture is the collection of several views.
3. **Node** – A node is a physically present sub-driving component that stands in for a computational resource.

**7.10 Assignment :**

1. Explain the principle of modelling

**7.11 Activity :**

1. Study the building blocks and UML.

**7.12 Case Study :**

1. Study the architecture of UML and summarize the five views of it.

**7.13 Further Readings :**

1. UML Distilled by Martin fowler, Third Edition
2. UML for java programmers by Robert C. Martin

**UNIT STRUCTURE**

- 8.0 Learning Objectives
- 8.1 Introduction
- 8.2 Classes
- 8.3 Relation
- 8.4 Common Mechanism
- 8.5 Diagrams
- 8.6 Advanced Classes
- 8.7 Advanced Relationship
- 8.8 Interfaces, Types and Roles
- 8.9 Packages
- 8.10 Let Us Sum Up
- 8.11 Answers for Check Your Progress
- 8.12 Glossary
- 8.13 Assignment
- 8.14 Activities
- 8.15 Case Study
- 8.16 Further Readings

**8.0 Learning Objectives :**

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

- About class and advanced classes in detail
- About Relationship and Advanced Relationship in Modelling
- About the common mechanism in Modelling
- About the different diagrams used in the Modelling
- About interfaces, types of role and packages in Modelling

**8.1 Introduction :**

The structural or conceptual model describes structure of objects that support in the business processes of an organization. During the analysis , the structural model presents the logical organization of objects without specifying how they are stored , created or manipulated so that focus on the company without being distracted by the technical details. Later, during design, the structural model is updated to reflect exactly how objects will be stored in databases and files. This unit describes the basic and advanced structure Modelling in detail.

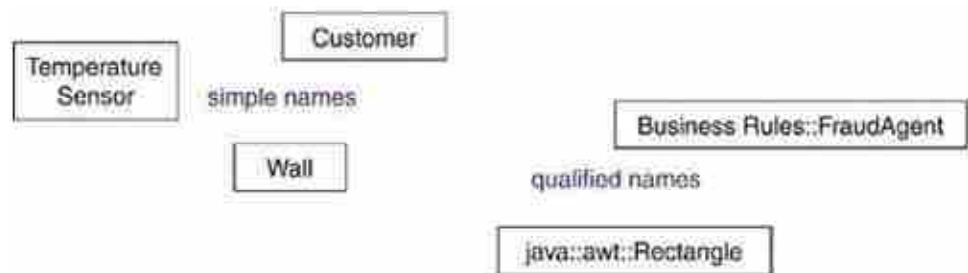
## 8.2 Classes :

### Terms and Concepts :

A group of objects with similar qualities, relationships, operations, and semantics are described as belonging to a class. A graphic representation of a class is a rectangle.

- **Names :**

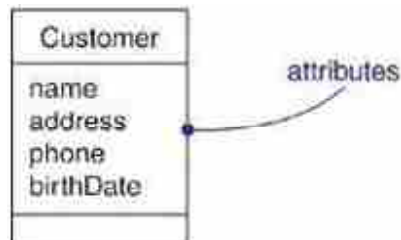
Every class needs a name that sets it apart from other classes. A name is a string of text. Only the simple form of that name is used; a qualified name is the class name followed by the name of the package the class is found in. A class's name alone can be used to draw it.



**Figure 8.1 : Names of class**

- **Attributes :**

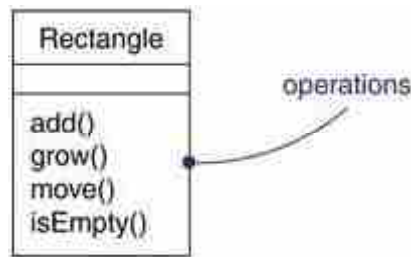
A named property of a class that specifies a range of values that instances of the property may have been called an attribute. A class may have one characteristic, several attributes, or none at all. A property of what you are modelling that is common by all objects in that class is represented by an attribute. You can model your consumers so that each has their name, address, phone number, and date of birth, just like each wall has a height, width, and thickness.



**Figure 8.2 : Attributes of class**

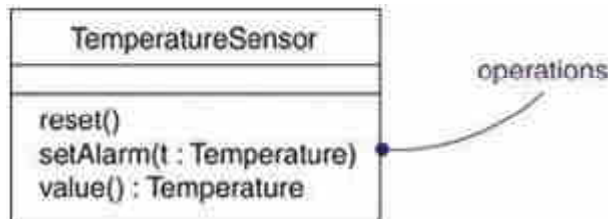
- **Operations :**

Any object in the class may be asked to provide a service known as an operation in order to change its behaviour. In other words, an operation is an abstraction of a shared feature across all members of a class of objects that you may accomplish with an object. Any number of operations, or none at all, can be included in a class. For instance, all objects belonging to the Rectangle class in a window library like the one in the Java awt package can be moved, resized, or queried. Invoking an operation on an object frequently (but not always) modifies the object's data or state. The processes are depicted graphically below the class properties of a room. Drawing operations using only their names is possible.



**Figure 8.3 : Operations of class**

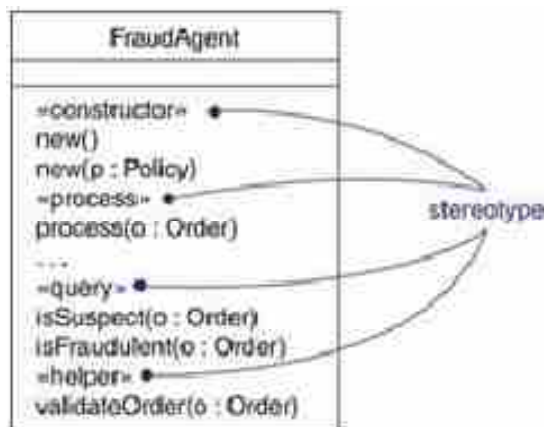
A function's signature, which contains the name, type, and list of values for each parameter as well as the return type, can be used to specify an operation.



**Operation of Class-2**

- Organization of attributes and operations :**

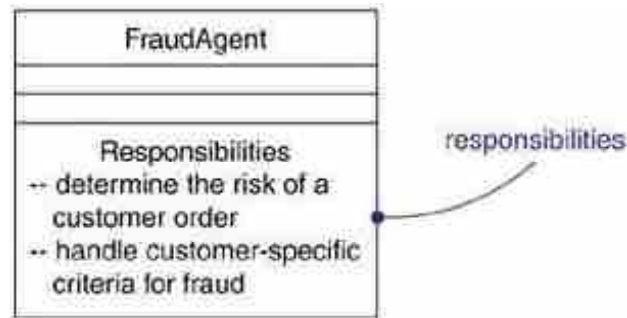
It's not necessary to show every attribute and every operation when drawing a class. In truth, you can't (there are too many to fit into a single figure) and probably shouldn't in the majority of circumstances (only a subset of these attributes and operations are likely relevant to a specific view). For these reasons, you can choose to reveal simply some or none of a class characteristics and operations by bypassing it. By adding an ellipsis at the end of each list, you can indicate that there are more traits or properties present than what is visible ("...").



**Figure 8.4 : Organization of attributes and operations**

- Responsibility :**

A duty is an agreement or a group obligation. All objects in a class are said to have the same type of state and behaviour when the class is created. These characteristics and related operations are merely the means through which the duties of the class are carried out on a more abstract level. A wall class is in charge of knowing the dimensions of height, width, and thickness; a fraud agent class, similar to the one found in a credit card application, is in charge of processing orders and determining whether they are legitimate, suspicious, or fraudulent; a temperature sensor class is in charge of measuring the temperature and sounding an alarm if the temperature reaches a specific level.



**Figure 8.5 : Representation of Responsibility of class**

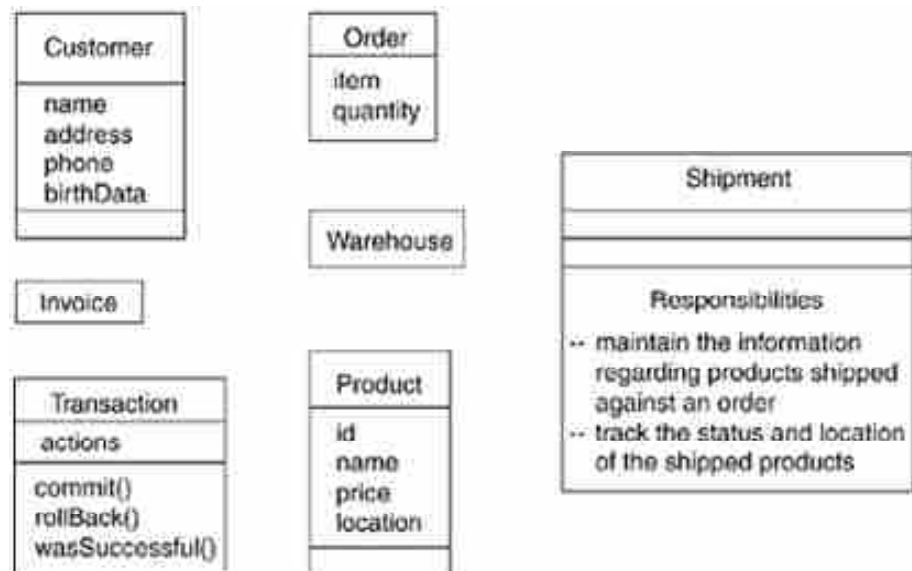
➤ **Common Modelling techniques :**

• **Modelling a system's vocabulary :**

Classes will be used most frequently to model abstractions taken from the current issue. You're attempting to resolve the issue, or the technologies you employ to do so, since each of these abstractions is a component of the language of your system, they collectively stand in for the elements that users and implementers value.

To model the vocabulary in a system,

- Define the terms used by users or implementers to describe the issue or the solution.
- Use case-based analysis and CRC maps to assist locate these abstractions.
- Determine a group of duties for each abstraction. Ascertain that each class has a distinct purpose and that the duties assigned to each class are distributed fairly.
- Define the properties and operations necessary for each class to carry out these functions.



**Figure 8.6 : Vocabulary of system**

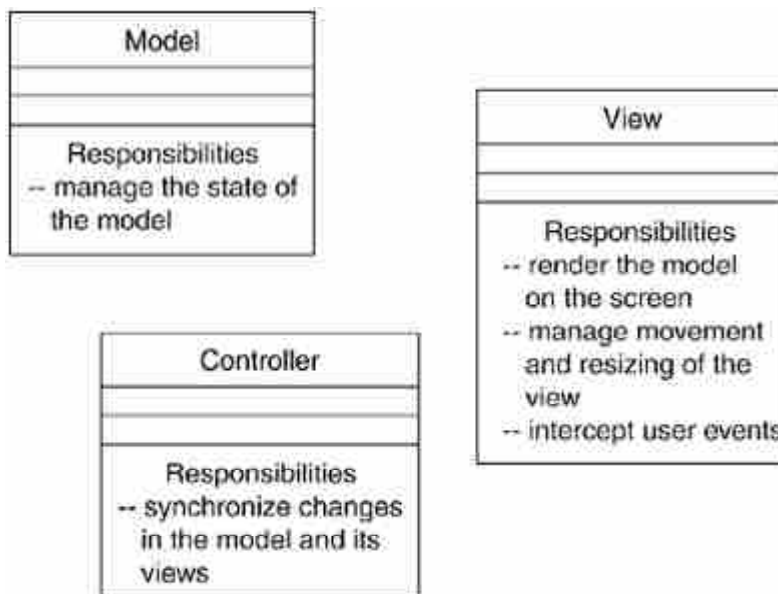
A collection of classes, such as customer, order, and product, that were taken from a retail system. Other relevant abstractions from the problem's language are shown in this image, including Shipping (used to track orders), Invoice (used to invoice orders), and Inventory (where products are stored). accessible prior to dispatch). The solution also has an abstraction called a transaction that applies to both orders and shipments.

- **Modelling the responsibilities distribution in a system :**

When you start Modelling more than a handful of classes, you want to be sure that yours abstractions give one balanced set of responsibility.

To model that distribution of responsibility in one system,

- Find a group of classes that collaborate closely to carry out a specific behaviour.
- Determine one group to be in charge of each of these classes.
- Take a look at this collection of classes as a whole, separate the classes that have too many responsibilities into smaller abstractions, combine tiny classes that have unnecessary duties in the elderly, and Y reassign responsibility for each reasonable abstraction that is based on its own.
- In order to ensure that no class in cooperation performs too much or too little, take into account how these classes collaborate with one another and disperse their duties properly.



**Figure 8.7 : Responsibilities distribution**

- **Modelling Things that are not software :**

To model not software stuff,

- Model that stuffs you are abstracting as a class.
- To distinguish these things from UML's defined building blocks, create one new building blocks by using stereotypes to specify this new semantics and give one distinctiveness visual aids.
- If what you are Modelling is a kind of hardware that contains software in itself, also consider Modelling it as a kind of node so you can expand your further structure.



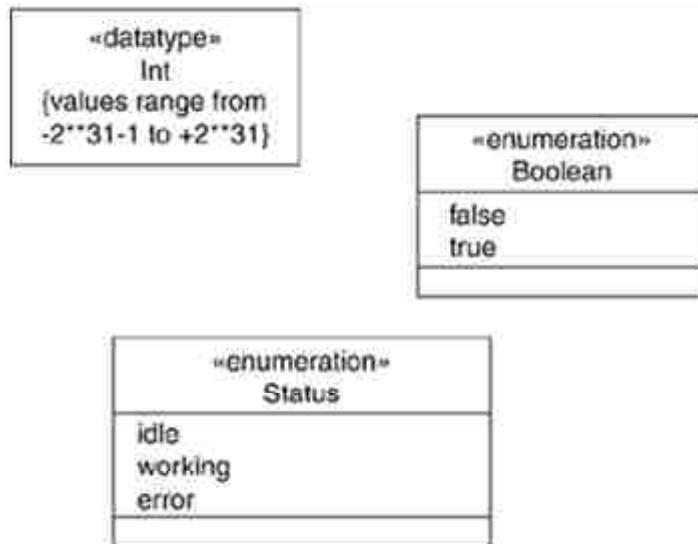
**Figure 8.8 : Modeling the thing that are not software**



➤ **Modelling of primitive types :**

To model primitive types,

- Represent the objects you are abstracting as a class or an enumeration, expressed with the appropriate stereotype in class notation.
- Use limits to provide a range of values from this type if you need to (constraints).



**Figure 8.9 : Modelling the primitive types**

❑ **Check Your Progress – 1 :**

1. Which are from following abstractions a class consists ?
  - a. Operations
  - b. Set of objects
  - c. Attributes
  - d. All of these

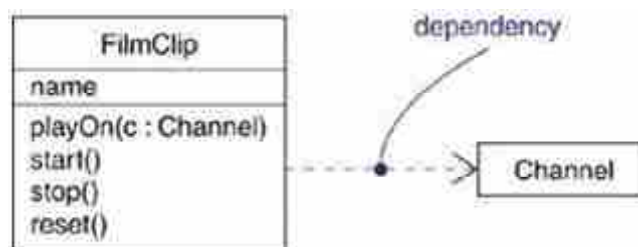
**8.3 Relation :**

**Terms and Concepts :**

A connection between two things is a relationship. Dependencies, connections, and generalisations are the three most crucial relationships in object-oriented modelling. A path is used to indicate a relationship, with various types of lines used to denote different types of relationships.

**Dependencies :**

A dependence is a relationship in which one object (like the Windows class) depends on another thing (like the Event class) for data and services, but not necessarily the other way around. A graphic representation of an addition is a dotted line that points in the direction of the dependant object. When you wish to display one thing while using another, choose dependencies.



**Figure 8.10 : Dependency**

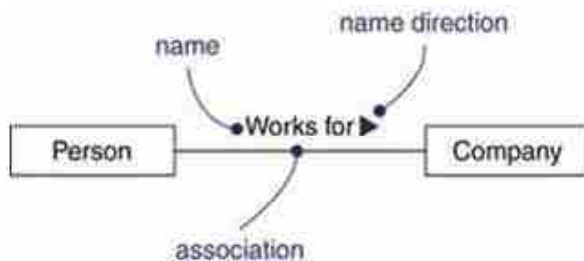
**Associations :**

An association is a structural relationship that indicates how two things' objects are related to one another. You can relate objects from one class to things from the other if you know the association between the two classes. Returning to the same class from either end of an association circle is perfectly permitted. It can therefore bind to other objects of the same class given an object from the class. Binary associations connect exactly two classes and are the most common type of association. You could have n-honour relationships, which are associations between more than two classes but are less typical.

In addition to this basic form, there are four decorations that apply to associations.

**Name :**

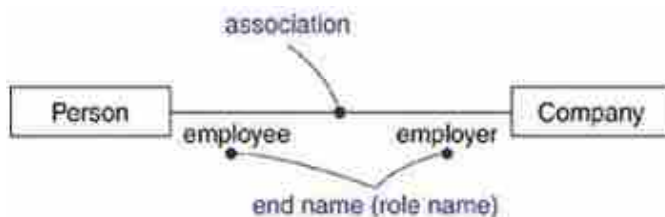
A name for an association can be used to identify the type of connection between the parties. You can give a name a direction by specifying a direction triangle that points in the direction you want to interpret the name in order to ensure that there is no doubt regarding its meaning.



**Figure 8.11 : Name–Associations**

**Role :**

A role is essentially the face that the class at the other end of the association presents to the class at the proximal end of the association. When a class participates in an association, it has a certain role to play in that relationship. The function a class performs within an association may be specifically mentioned. The endpoint name refers to the function that an association's endpoints perform (in UML1 it was called the role name). The business class, which assumes the position of the employer, is connected to the person class, which assumes the function of the employee.



**Figure 8.12 : Roles–Association**

**Multiplicity :**

A structural relationship between items is represented by an association. It is crucial to indicate how many objects can be connected through a particular instance of an attachment in many modelling circumstances. The diversity of a role played by an association is the "how many" in question. It displays an integer range that identifies the potential size of the set of connected items.

## Object Oriented Analysis and Design

The number of objects must be within the given range.

- It can display.. a multiplicity of exactly one (1),
- zero or one (0..1),
- many (0..\*) or two or more (2..\*).
- Interval of integers (as 3..6 ).
- You can even enter an exact number (for example, 4, which is equal to 4..4).

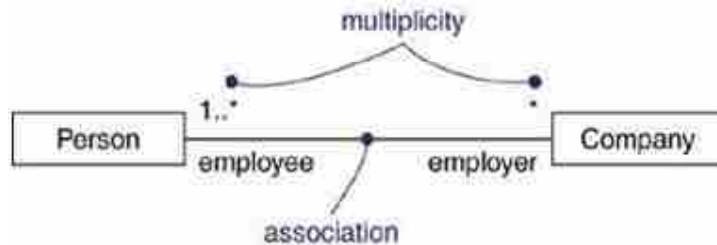


Figure 8.13 : Multiplicity–Associations

### Aggregation :

An association between items depicts a structural relationship between them. In many modelling scenarios, it is critical to specify how many objects can be connected via a certain instance of an attachment. The "how many" in question depends on the variety of roles that an association fills. It indicates the potential size of the collection of connected objects with an integer range.

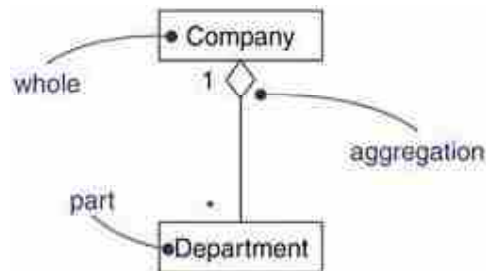
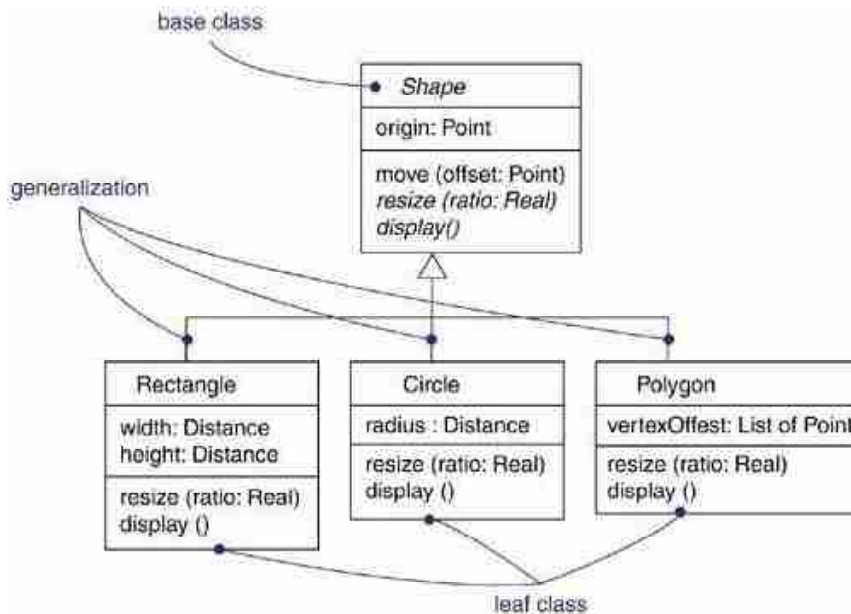


Figure 8.14 : Aggregations

### Generalizations :

A generalisation is a connection between a broad category of something (also known as a parent category or superclass) and a more focused category of somethings (called a child or subclass). A relationship where a thing (like the Bay Window class) is a type of something more general is known as generalisation (such as the Windows class). A variable or parameter written by the parent class cannot be used for an object in the child class and vice versa.



**Figure 8.15 : Generalizations**

❑ **Check Your Progress – 2 :**

1. The overall \_\_\_\_\_ of the system is described during the design phase.
  - a. System flow
  - b. Data Flow
  - c. Architecture
  - d. None of these

**8.4 Common Mechanism :**

**Terms and Concepts :**

A **note** is a visual representation of restrictions or comments related to an element or group of elements. A graphic note includes a textual or graphic commentary and is displayed as a rectangle with one corner pointing up.

To construct new sorts of building blocks that are similar to pre-existing ones but particular to your problem, you can extend the UML vocabulary by using stereotypes. Graphically, a **stereotype** is shown as a name placed above the name of another element and encircled in guillemets (French quotation marks of the form "").

A new icon connected to the stereotype might also be used to indicate the stereotypical element.

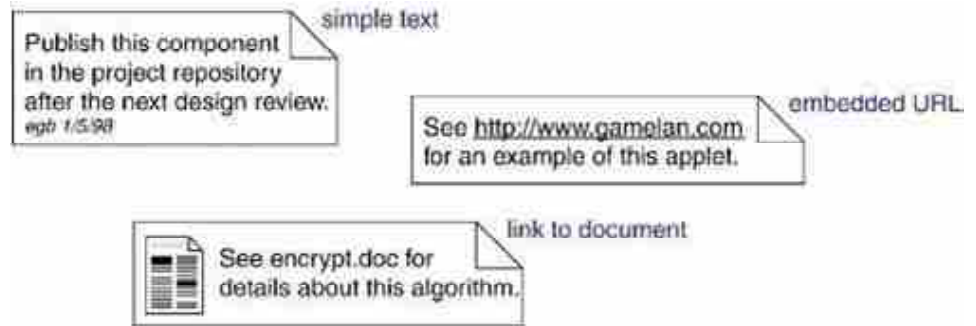
A stereotype's **tagged value** is a feature that enables you to add new data to an element that shares that stereotype. A string of the form name = value in a note linked to the object serves as the graphic representation of a labelled value.

A **constraint** is a textual description of a UML element's semantics that enables you to add new rules or change old ones. A constraint is shown graphically as a string enclosed in square brackets and is either present next to the related element or is linked to the associated element or elements via dependency relationships. As an alternative, a constraint can be represented in a note.

• **Notes :**

A note that includes a commentary has no semantic effect, which indicates that its content has no impact on the model's interpretation. Due to representational constraints, notes are utilised to specify needs, remarks, modifications, and explanations.

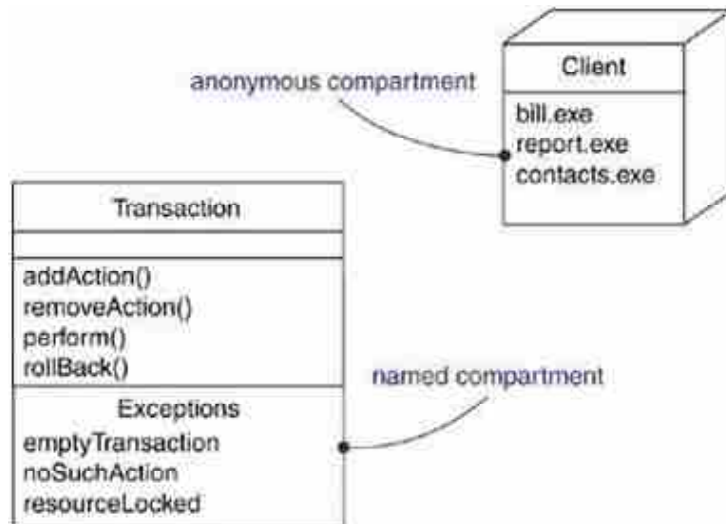
A note can contain any combination of text or graphics.



**Figure 8.16 : Notes**

**Other Adornments :**

Adornments are textual or graphic elements that are added to the basic notation of an element and are used to show details about the specification of the element.

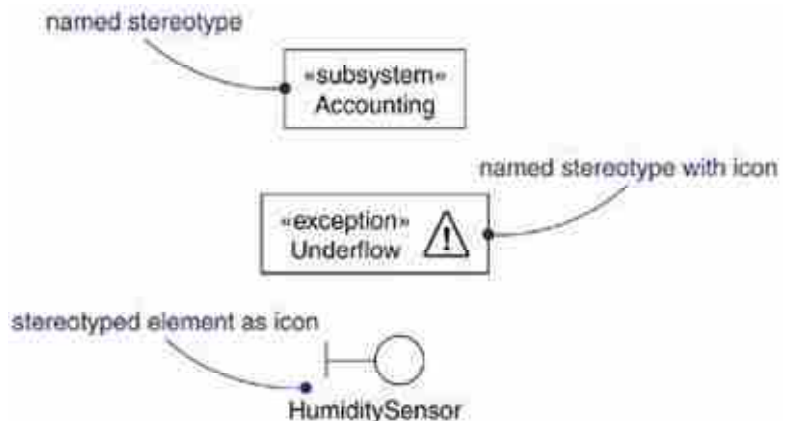


**Figure 8.17 : Other Adornments**

- Stereotypes :**

The UML offers a language for notational, grouping, behavioural, and structural concepts. The great majority of the systems you need to represent are covered by these four fundamental sorts of objects.

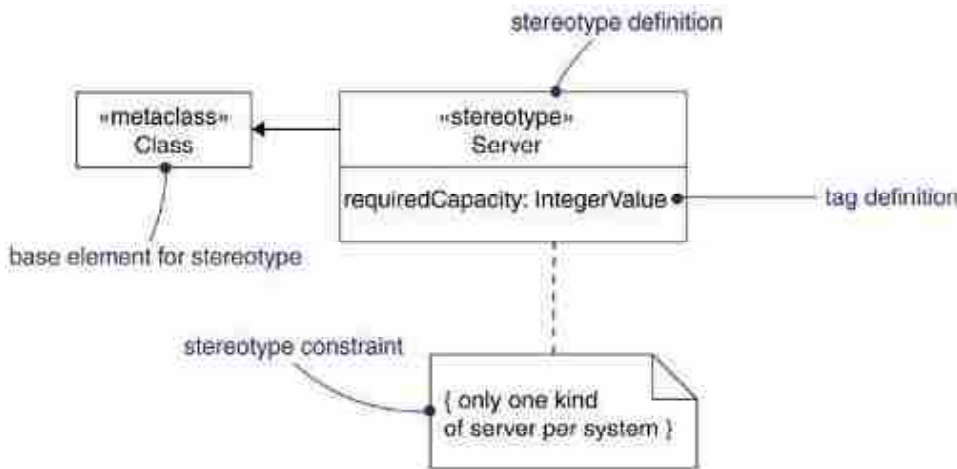
In its most basic form, a stereotype is expressed as the name of another item placed on top of "Name" among the guillemets.



**Figure 8.18 : Stereotypes**

• **Tagged Values :**

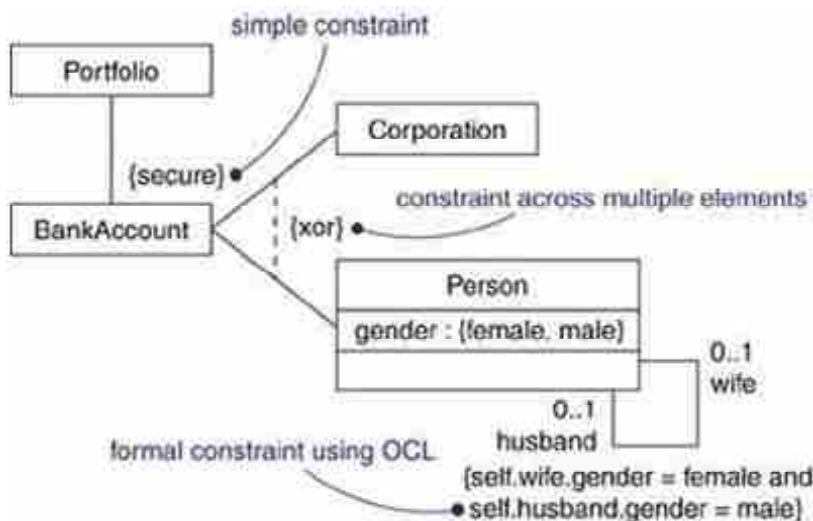
Every element in UML has its own unique set of properties, including associations, classes, and classes' names, attributes, and operations. To UML, stereotypes can be used to add new elements, and tagged values can be used to add new properties to a stereotype.



**Figure 8.19 : Tagged Values**

• **Constraints :**

In UML, each element has a distinct semantics. The Liskov substitution principle is involved in generalisation (typically if you know what is best for you), and several connections with a class point to unique relationships. You can modify current rules or add new interpretations with some limitations. The requirements that a runtime configuration must meet in order to suit the model are specified by constraints.



**Figure 8.20 : Constraints**

stereotype specifies that the classification is one stereotype there can be used for other items

❑ **Check Your Progress – 3 :**

1. Which concepts from the following hide the internal implementation of an object ?
  - a. Inheritance
  - b. Polymorphism
  - c. Encapsulation
  - d. Abstraction

## **8.5 Diagrams :**

A **diagram** is a visual representation of a collection of items. Vertices (things) and arcs are typically joined to form connected graphs (relations).

Regardless of the problem domain, you will create the same types of diagrams in Modelling actual systems because they show similar viewpoints on common models. The static components of a system are typically represented by one of the following diagrams.

1. Class diagram
2. Object diagram
3. Component diagram
4. Composite structure diagram
5. Deployment diagram
6. Artifact diagram

Frequently, you'll require five more charts to see a system's dynamic components.

1. Use case diagram
2. Sequence diagram
3. State diagram
4. Activity diagram
5. Communication diagram

- **Structural diagrams**

The static components of a system can be visualised, specified, built, and documented using UML structural diagrams. A system's relatively stable skeleton and support structure can be visualised as its static components. The existence and location of elements like classes, interfaces, collaborations, components, and nodes are all examples of static aspects of a software system, just as the static aspects of a house include the existence and location of elements like walls, doors, windows, pipes, cables, and ventilation openings.

The basic sets of elements you will come across when modelling a system are roughly where UML structural diagrams are arranged.

1. Class diagram:	class, interfaces, and collaborations
2. Object diagram:	Objects
3. Component diagram:	Components
4. Implement diagram:	nodes

- **Behavioural Diagrams :**

The dynamic components of a system are seen, specified, built, and documented using UML behaviour. The dynamic features of a system can be thought of as its evolving components. The dynamic elements of a software system also include things like the flow of messages through time and the physical movement of components over a network, just as the dynamic elements of a building include the flow of air and traffic through the areas of a house.

The most significant approaches to model a system's dynamics are essentially the divisions around which UML behavioural charts are arranged.

1. Use case diagram	Organize that behaviour from that system
2. Sequence diagram	Spotlights on that time request from messages
3. Stat diagram	Spotlights in the change Express from one system driven to events
4. Activity diagram	Focusing on flow activity monitoring to the activity
5. Collaboration diagram	Spotlights on that structural organization from objects to send and receive messages

**❑ Check Your Progress – 4 :**

1. \_\_\_\_\_ diagram has a static view from the following list.  
 a. class            b. Activity            c. Object            d. Use case

**8.6 Advanced Classes :**

**Terms and Concepts :**

A method that defines structural and behavioural features is a classifier. Classes, relationships, data types, interfaces, signals, nodes, components, nodes, use cases, and subsystems are all examples of classifiers.

**Classifiers :**

When you model, you will find abstractions that reflect both the elements of your solution and the real world. For instance, your projector inventory will probably include a customer class (representing people placing orders for products) and a transaction class if you are developing a web-based ordering system (an implementation artefact representing an action). atomic). You might have a price element in the implemented system where instances are located on each client node. Separating the essence and appearance of objects in your environment is a key component of modelling. Each of these abstractions will have examples.

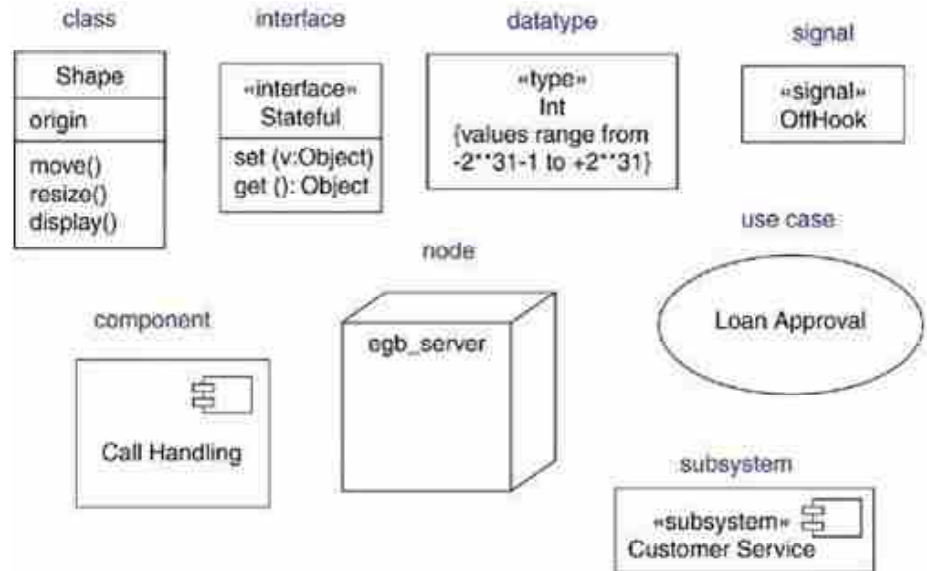
The class is the most significant kind of classifier in UML. A group of objects with similar qualities, operations, relations, and semantics are described as belonging to a class. Classes are not the only kind of classifier, though. You can model using a variety of additional classifiers included in the UML.

Interface	A collection of operations that are used to specify a Service from one class or either one component
Datatype	A type whose values is immutable, included primitive Incorporated types (such as string and number) as well as enum types (i.e boolean)
Association	A description of one set of links, each of which relates to either two or more objects.
Signal	The specification from one asynchronous communicated message Between instances
Component	A modular part from one system to hides his implementation bag one place from external interfaces



**Object Oriented  
Analysis and Design**

Note	A physical element to exists in to run time and that represents one computationally resource, in general have at least some memory and often processing ability
Use case	A description of a set of sequence of action, including variants, to one system do to produces one observable Result from value to a particular actor
Subsystem	A component representing a major part of a system

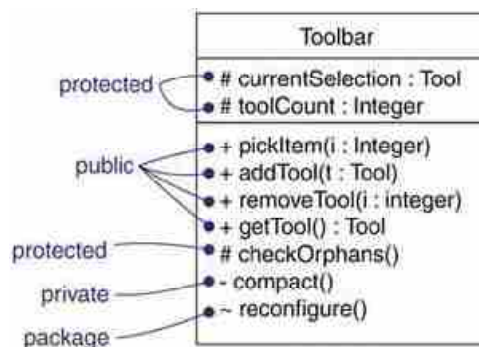


**Figure 8.21 : Classifier**

**Visibility :**

Visibility is one of the design elements you can define for a property or operation. If a function is visible, it means that other classifiers can utilise it. You can declare up to four levels of visibility in UML.

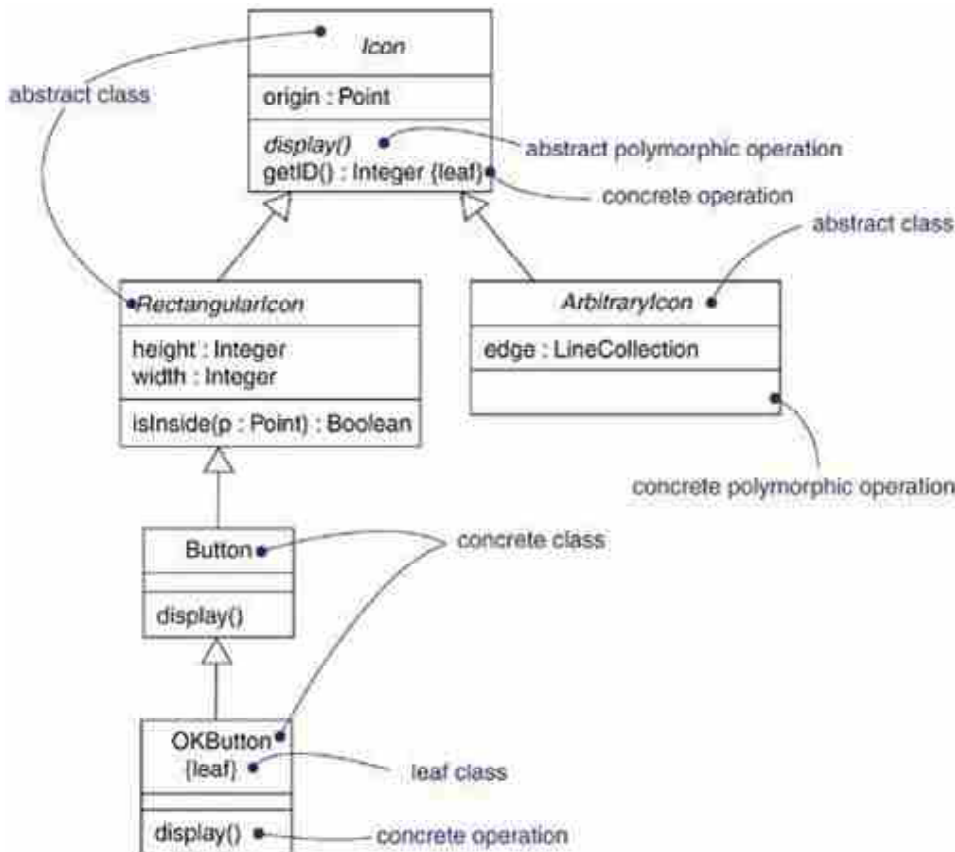
1. public	anyone outside sorting with visibility to that Cube sorting able to wear that function; specified to put before that + symbol.
2. protected	Any descendant from that sorting able to wear that function; specified to put before that symbol #.
3. private	Only that The classification itself can use the characteristics; specified by put before that symbol -.
4. package	Only classifiers declared in that same package able to wear that function; specified to put before ~ symbol.



**Figure 8.22 : Visibility**

**Abstract, leaf and polymorphic elements :**

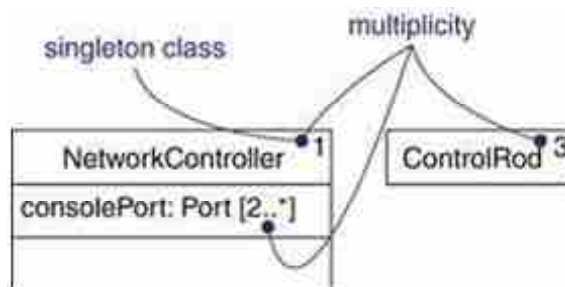
One lattice from classes is modelled using generalisation relationships, with more general abstractions at the top and more specific ones at the bottom. It is common practise to state in these hierarchies that some classes are abstract, which necessitates the requirement that they lack direct cases. You can indicate an abstract class in UML by italicising its name. These abstract classes include Icon, RectangularIcon, and ArbitraryIcon. Instead, a grade that is concrete (like the What and OK buttons) can have direct examples.



**Figure 8.23 : Abstract, leaf and polymorphic elements**

**Multiplicity :**

It is fair to presume that a class can have any number of instances when using it. as a grade (unless, of course, to it is an abstract class could not have someone directly instances, though over there able to be none number of instances from it is concrete children).



**Figure 8.24 : Multiplicity**

**Attributes :**

When modelling a class' structural features (i.e., its attributes), you simply write the name of each attribute at the most abstract level.

```
visibility ] Name
[':' type] ['(
multiplicity ] ')] ['='
initial value]
[ string property {',' string-property}]
```

To example, monitoring is all legal attribute declarations :

origin	name only
+ origin	Visibility and Name
origin: Point	Name and type
Name: String [0..1]	Name, type, and diversity/multiplicity
origin: Point = (0,0)	Name, type, and initial value
ID: Integer {read-only}	Name and property

**Operations :**

while simulating a class's behavioural traits at the most abstract level. Additionally, each operation allows you to specify arguments, return types, simultaneity semantics, and a variety of other features. Overall, an operation's name is plus its parameters (including its return type, if any). referred to as the operation's signature.

```
[ visibility ] name ['( parameter list ')'] [':' return type]
[ string property {',' string-property}]
```

To example, that following is all valid operation statements :

display	name only
+ display	Visibility and Name
set : (n: Name, s : String)	Name and parameters
getID () : Integer	Name and return type
restart () {guarded}	Name and property

**8.7 Advanced Relationship :**

**Terms and Concepts :**

**Relationship :**

A **relationship** it is one Connection Come in stuff. Dependencies, affiliations, generalisations, and realisations are the four key relationships in object-oriented modelling. A relationship is graphically represented as a path, with various types of lines being utilised to demarcate the various relationships.

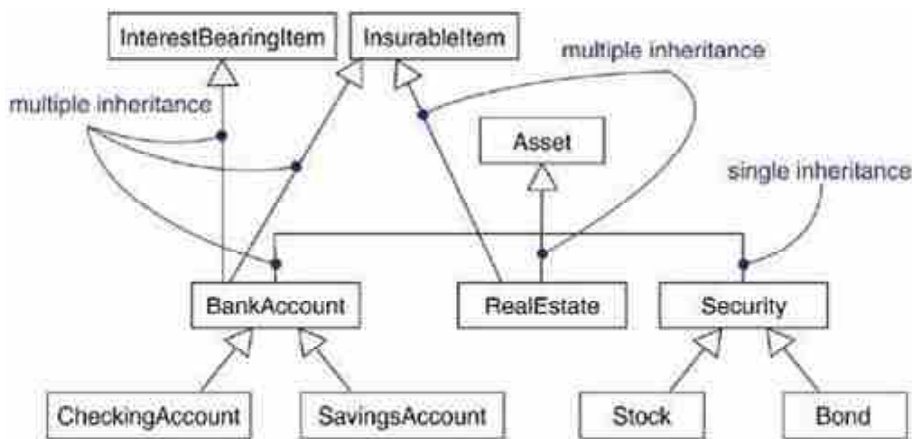
**Dependencies :**

**Dependency** is a usage relationship that shows how a change in the specification of one thing (like the SetTopController class) may affect another thing (like the class Channellterator) that uses it, but not the other way around. Graphically, an addiction is represented as a dotted line, aimed at what it depends on. Use dependencies when you want to show something using other things. A simple, unadulterated relationship of dependence it is enough to the majority of use relationships however, you will find yourself if wants to specify

a nuance of meaning, the UML defines one range of stereotypes that can be applied to additive relationships.

**Generalizations :**

A **generalisation** is a connection between a parent class (sometimes known as a superclass) and a more specialised class (called the child or subclass). For instance, try looking for the general class Window and its particular subclass, multi-pane window. The child (MultiPaneWindow) will inherit the parent's entire structure and behaviour with just one generalisation relationship from child to parent (window).



**Figure 8.25 : Generalization**

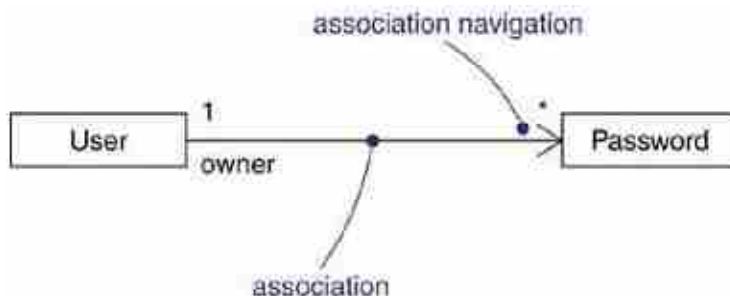
A plain, unmade generalisation relationship it is enough to the majority from that inheritance relationships you find.

**Associations :**

A structural relationship called an **association** shows that one object's objects are linked to those of other persons. For instance, a book might have a one-to-many relationship with a library class. grade, which shows that each instance of a book belongs to one instance of a library.

**Navigation :**

It is possible to move from items of one kind to objects of the other type given a simple, unmade relationship between two classes, such as Book and Library. Unless otherwise mentioned, there are two ways to navigate through an association.

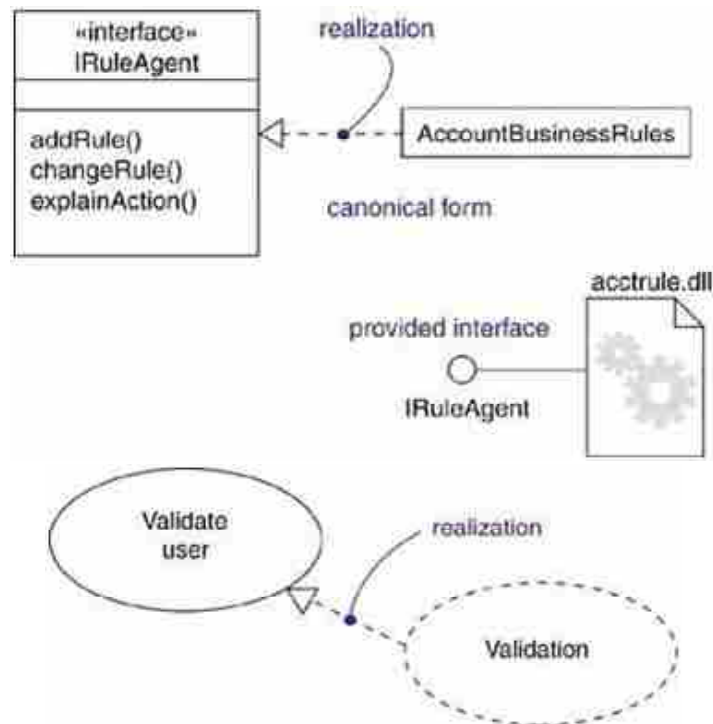


**Figure 8.26 : Navigation**

**Realizations :**

In a realisation, two classifiers have a semantic relationship in which one classifier specifies a contract that the other classifier promises to fulfil. A cognition is graphically depicted as a single directed dotted line with a sizable open arrowhead pointing in the direction of the categorization denoting contract.

Relationships based on realisation are sufficiently distinct from those based on addition, generalisation, and association to be classified as a distinct relationship type. Cognitively speaking, there is a form of crossover between addition and generalisation, and its notation combines notation to reliance and generalisation.



**Figure 8.27 : Realizations**

**8.8 Interfaces, Types and Roles :**

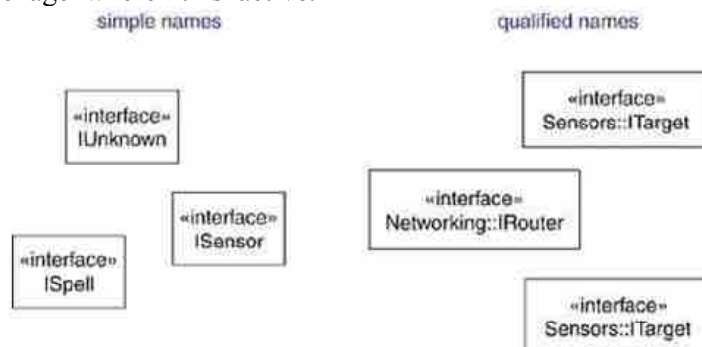
**Terms and concepts :**

A service of a class or component is specified by an interface, which is a group of activities. A type is a stereotype from one grade that is used to define an object's domain and the operations (but not the methods) that are applicable to it. A paper is the actions of a unit taking part in a certain environment.

To disclose his operations and other attributes, one Interface can be displayed graphically as a stereotype class.

**Names :**

Each interface needs a name that sets it apart from the others. A **name** is a text string that only refers to an acquaintance by one straightforward name or one path. The name of the interface is the name that comes before the name of the package where it is active.



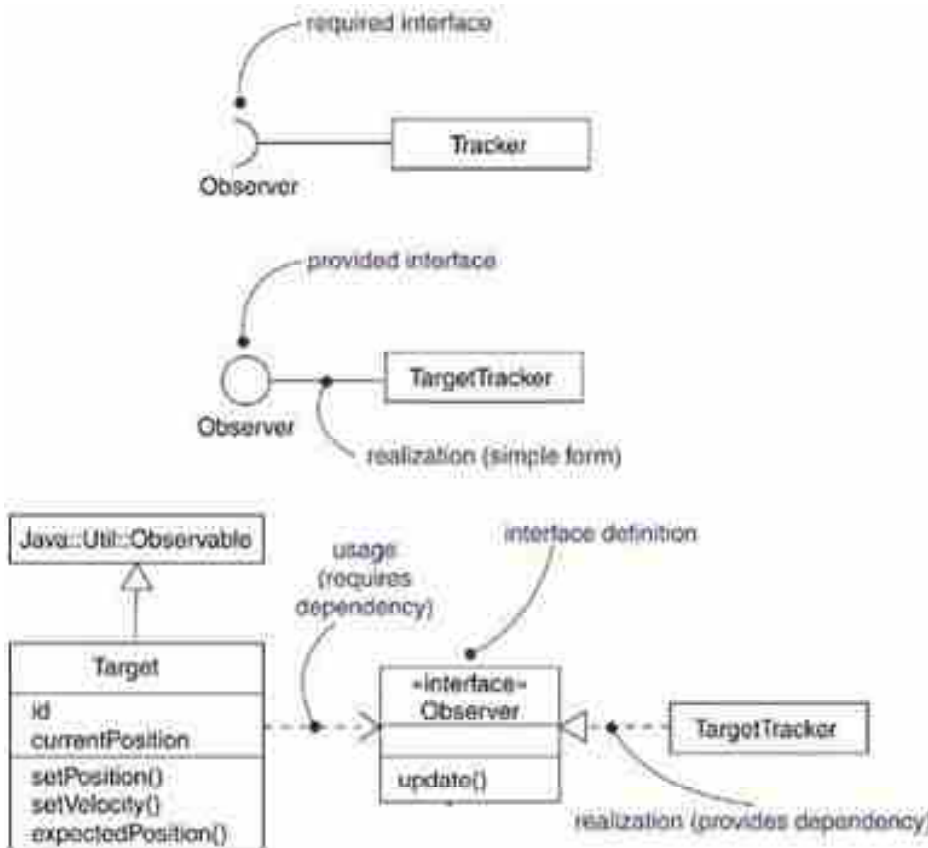
**Figure 8.28 : Names**

**Operations :**

A service of a class or one component is specified by an interface, which is a named collection of operations. Interfaces, in contrast to classes or types, do not specify any implementation (so they can be), and they do not contain any methods that permit the performance of an action. The interface can have any number of actions because it is a class. These procedures can be enhanced using constraints, labelling values, stereotypes, visibility properties, and concurrency properties.

**Relationships :**

It is a semantic link between two classifiers, where one classifier sets a contract that other stuff classifier guarantees to carry outside. Like a class, an interface is capable of participating in generalisation, association, and dependent relationships in addition.



**Figure 8.29 : Relationships**

**Understanding an Interface :**

The first thing you see when given an interface is a set of operations that describe one service of a class or component. If you go a little deeper, you'll discover each operation's whole signature as well as any unique attributes, such as visibility, scope, and concurrent semantics.

**8.9 Packages :**

**Terms and Concepts :**

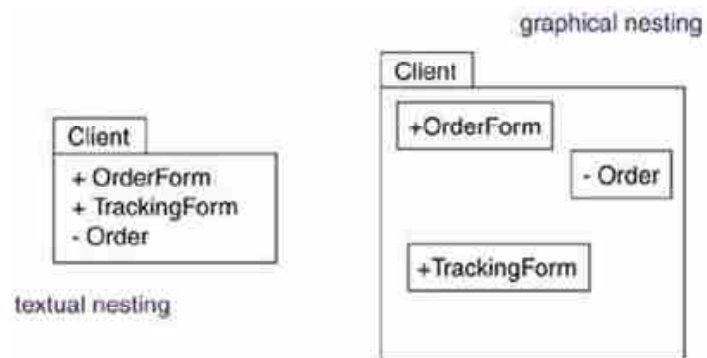
A **package** is a general mechanism to organize the model itself into a hierarchy; It has no means to the execution. Graphically, one package is rendered with one eyelash file. That the package name goes in the folder (if the contents are not displayed) or in the tab (if the contents are from the folder is see you).

**Names :**

Every package needs a **name** that sets it apart from other packages. A name is a string of text. Only the simple form of that name is known; a qualified name is made up of the name of the package, followed, if applicable, by the name of the package in which the package resides. Coordinated package names with a double colon (::)

**Owned Elements :**

Classes, interfaces, components, nodes, collaboration, use cases, diagrams, and even other packages are all contained in a package that can own other commodities. Due to the fact that ownership is a composite connection, the element is declared in the package. The element is destroyed if that package is destroyed. Every element is a package's exact property.



**Figure 8.30 : Owned Elements**

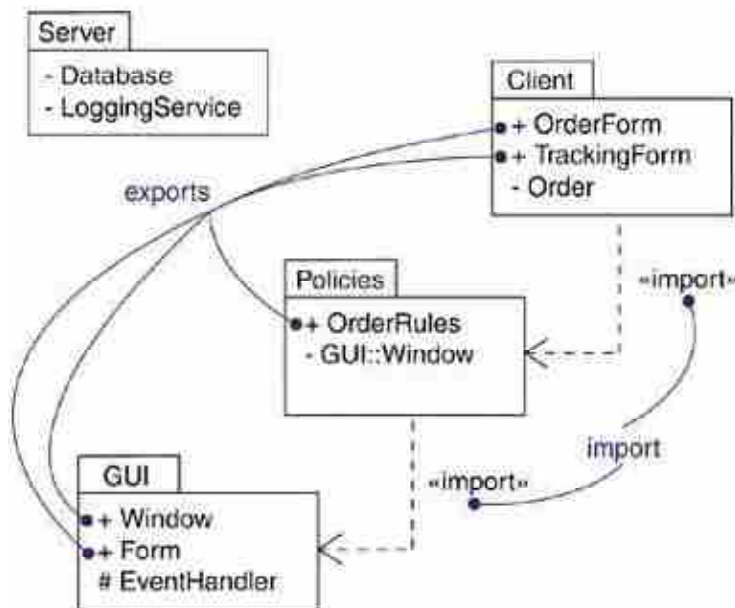
**Visibility :**

Similar to how you may verify the visibility of the properties and actions owned by a class, you can check the visibility of items that belong to a package. An item is often owned by one whose package is public, meaning that any package importing the package can see its contents. envelope-package item In contrast, only children can see protected items, and although private items are declared, they cannot be seen outside the package.

**Importing and Exporting :**

Instead, imagine that you placed A in one package and B in another, and that both packages were placed next to one another. This is a completely different scenario if A and B are also declared public sections of their respective packages. Despite A and B being open to the public, you can access one of the classes in the second package without providing a qualified name. However, if A's bundle imports B's package, B can no longer see A without a Name but A may now see B directly.

The target package's public elements are added to the import package's public namespace by the import. You model an import relationship in UML as an addiction embellished with the import stereotype. You can manage the complexity of long speak up from abstractions by packaging his abstractions in considerably bidder and controlling his access to import.



**Figure 8.31 : Importing and Exporting**

**❑ Check Your Progress – 5 :**

1. Select most suitable answer for class.
  - a. Class is a template for objects of a particular type
  - b. Class is a class of objects
  - c. Class is a classification of objects
  - d. Class is a group of objects

**8.10 Let Us Sum Up :**

In this unit you learnt about class in depth with its property and functionality. Also, we learnt different types of relationship in Modelling. You also learnt the common mechanism in Modelling; you have also learned different diagram which we can make for system in Modelling. You also learn the advanced classes and advanced relationship for Modelling. You also learn the interfaces, types of roles and packages with diagram in Modelling.

**8.11 Answers for Check Your Progress :**

- ❑ **Check Your Progress 1 :**  
1 : d
- ❑ **Check Your Progress 2 :**  
1 : c
- ❑ **Check Your Progress 3 :**  
1 : c
- ❑ **Check Your Progress 4 :**  
1 : d
- ❑ **Check Your Progress 5 :**  
1 : a



**8.12 Glossary :**

1. **Attribute** – A named property of a class that specifies the range of values that instances of the property might have been called an attribute.
2. **Dependencies** – Dependency is a connection where one item uses the knowledge and resources of another entity, but not always the other way around.

**8.13 Assignment :**

1. Explain the different parts of deployment diagram in UML.

**8.14 Activities :**

1. Discuss the different views that have to be considered before the build-up of an object-oriented software system.

**8.15 Case Study :**

1. Discuss the Interfaces, Types and Roles in detail.

**8.16 Further Readings :**

1. UML Distilled by Martin Fowler, Third Edition
2. UML for Java programmers by Robert C. Martin

**UNIT STRUCTURE**

- 9.0 Learning Objectives
- 9.1 Introduction
- 9.2 Class Diagrams : Terms & Concepts
- 9.3 Modelling Techniques for Class Diagrams
- 9.4 Object Diagrams : Terms and Concepts
- 9.5 Modelling Techniques for Object Diagrams
- 9.6 Let Us Sum Up
- 9.7 Answer to Check Your Progress
- 9.8 Glossary
- 9.9 Assignment
- 9.10 Activities
- 9.11 Case Study
- 9.12 Further Readings

**9.0 Learning Objectives :**

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

- Study about terms & concepts of class diagram
- Study about Modelling techniques of class diagram
- Study about terms & concepts of object diagram
- Study about Modelling techniques of object diagram

**9.1 Introduction :**

Class diagrams are one of the main building blocks of object-oriented Modelling. They are the only UML charts that can be directly mapped to OOP languages like Python, Java and C ++. This makes them one of the most popular UML charts among software developers because they are useful during software design.

An object diagram can be referred to as a screenshot of the instances of a system and the relationship between them. Since object diagrams represent behaviour once objects have been instantiated, we can study the behavior of the system at a particular moment. Object diagrams are essential to represent and understand the functional requirements of a system.

In other words, "An object diagram in Unified Modelling Language (UML) is a diagram that shows a complete or partial view of the structure of a modelled system at a particular time".

An object diagram is similar to a class diagram, except that it shows the instances of the classes in the system. We represent real classifiers and their relationships using class diagrams. On the other hand, an object diagram

represents specific instances of classes and relationships between them at a given time.

## **9.2 Class Diagrams : Terms & Concepts :**

An illustration of a group of classes, interfaces, and collaborations along with their connections is called a class diagram. A class diagram visually consists of a group of vertices and arcs.

### **Common features :**

A class diagram is merely a particular kind of diagram that has the same name and graphic content that is a projection on a model as all other diagrams. A class chart's unique content distinguishes it from other kinds of charts.

### **Contents :**

Class diagrams Generally Contains the following stuff :

- Classes
- Collaborations
- Interfaces
- Dependence, association and generalization relationships

Just like all other diagrams, class diagrams able to contain notes and constraints.

Class diagrams that can also Contains either packages or subsystems, each of which is used to group the model's components into bigger chunks. You might occasionally want to include examples in his grade diagrams. What a shame, especially when you have to picture that (perhaps dynamic) type from a given instance.

### **Common uses :**

Class diagrams are used to represent a system's static design view. This viewpoint primarily supports the idea that a system's service providers should provide end users with the functional requirements for that system.

When you model that static design view of the system, you want to do it typical one of the diagrams from its three ways.

#### **1. To model the vocabulary of a system**

Choosing which abstractions fall within and beyond the bounds of a system is necessary for modelling its vocabulary. To describe these abstractions and their roles, utilise class diagrams.

#### **2. To model simple collaborations**

A community of classes, interfaces, and other components come together to form collaboration, which results in collaborative behaviours that are more effective than the combination of their individual parts. For instance, it is not enough to look at a single class to comprehend the semantics of a transaction in a distributed system. On the other hand, a group of classes working collectively performs this semantics. These categories and their relationships are depicted and specified using class diagrams.

#### **3. To model a logical database schema**

Consider a schema as a conceptual design model for a database. You wish to store durable data in a relational database or an object-oriented database

in many different domains. Class diagrams can be used to model the databases' schemas.

❑ **Check Your Progress – 1 :**

1. A class is divided into which of these parts from the following ?
  - a. Attribute part
  - b. Name part
  - c. Operation part
  - d. All of these

### 9.3 Modelling Techniques for Class Diagrams :

#### 1. Modelling of simple collaborations to model collaboration :

- Determine the mechanism you would use to do it; I prefer to simulate it. an apparatus It depicts some behaviour or function for the aspect of the system you are modelling that results from social interactions with classes, interfaces, and other elements.
- Determine the classes, interfaces, and other collaborations that are a part of each mechanism. Determine the relationships that also exist in these objects.
- To review these concepts, use situations. Along the way, you'll find that some of your model was lost, while other pieces were just semantically incorrect.
- Make sure to include your content in these fields. Get a decent class to start with to balance responsibility. After that, gradually transform these into actual attributes and actions.

#### 2. Modelling of a logical database schema :

To model a schema,

- Identify the classes in your model whose condition should exceed the life of your applications.
- Make a class diagram with these classes in it. For database-specific details, you can create your own collection of stereotypes and tagged values.
- Extend the classes' structural specifics. Generally speaking, this entails describing the specifics of their traits and concentrating on the relationships and diversity that pertain to these groupings.
- Recognize typical patterns, such cyclic relationships and one-to-one associations, that exacerbate physical database design. Make intermediate abstractions where necessary to make your logical structure simpler.
- How these classes behave when performing activities crucial for data access and data integrity. Business rules for managing collections of these objects should typically be contained in a layer above these persistent classes to improve the separation of concerns.
- Use technologies that can help you convert your logical design into a physical design whenever it's practical.

#### 3. Forward and reverse engineering :

The process of translating a model into code by mapping it into an implementation language is known as forward engineering. Because UML models are more semantically rich than models expressed in any other object-oriented programming language at the moment, forward engineering leads in information loss. In fact, this is a key factor in the necessity for models in

## Object Oriented Analysis and Design

addition to code. In contrast to raw code, structural features like collaborations and behavioural features like interactions may be seen clearly in UML.

To Forward an engineer a class diagram,

- Determine the association rules for the implementation language you have chosen (s). You want to carry out this action for the benefit of your project or your business as a whole.
- You might want to limit the use of some UML functions depending on the semantics of the languages you choose. For instance, while Smalltalk only supports a single inheritance, UML allows you to describe numerous inheritances. Instead of making your models linguistically reliant by prohibiting developers from using multiple inheritance, you might create idioms that would translate these richer features into the implementation language (making mapping more complex).
- Utilize tagged values to direct your target language's implementation choices. If you want to exercise fine-grained control, you can accomplish this at the grade level level. Additionally, you can do it at a more advanced level via collaborations or packages.
- Utilize software to create code.

```
public abstract class EventHandler1
{
    EventHandler1 successor1;
    private Integer currentEventID;
    private String source; EventHandler1() {}
    public void handleRequest1() {}
}
```

Reverse engineering is the process of translating code to a particular implementation language in order to turn it into a model. Reverse engineering generates a flow of data, some of which is less detailed than what is required to create usable models. However, reverse engineering is not yet complete. Engineering models lose information when they are converted to code, so it is impossible to fully reproduce a model from code unless your tools encode information in the source comments that is not confined to the implementation language's semantics.

To reverse engineer a class diagram,

- Determine the mapping rules for your implementation language or other language of preference. You want to carry out this action for the benefit of your project or your business as a whole.
- You can reverse a piece of code by pointing a tool at it. Use your tool to create a new model or change one that has already been designed. Reverse engineering a single clear model from a vast quantity of code is not something that can be done easily. You must choose a section of the code and construct the model from the ground up.

- Using your tool, query the model to produce a class diagram. For instance, you may start with one or more classes and then enlarge the diagram by tracing certain connections or incorporating nearby classes. As necessary to convey your intentions, reveal or conceal information about the items in this class chart.
- By hand–adding design information to the model, you can express design intentions that aren't expressed or are otherwise obscured by the code.

❑ **Check Your Progress – 2 :**

1. An operation can be described as which of the following ?
  - a. Class behavior
  - b. Object behavior
  - c. Function behavior
  - d. Class and object behavior

<b>9.4 Object Diagrams : Terms and Concepts :</b>
---

Class diagrams contain objects, which are represented by object diagrams. A group of objects and their connections are displayed simultaneously in an object diagram.

An object chart is a diagram that displays several items and their connections all at once. An object diagram visually consists of a group of vertices and arcs.

**Common Properties :**

An object diagram is a particular kind of diagram that has the same name and graphic content that is a projection on a model as all other diagrams. An object chart's unique content distinguishes it from all other chart kinds.

**Contents :**

Object charts commonly cover

- Objects
- Links

Like all other diagrams, an object diagram can include notes and limitations.

When you have visualised, the classes underlying each instance, you may want to add classes to your object charts from time to time.

**Common uses :**

Like class diagrams, object diagrams are used to represent the static design view or the static process view of a system from the viewpoint of actual or prototype instances. This perspective largely supports a system's functional requirements, or the services that the system must offer to its users. Static data structures can be modelled using object diagrams.

You commonly utilise object diagrams in one method when modelling the static design view or the static process view of a system:

**To model object structures :**

Object structure modelling It also comprises taking a single photo of those things in one system at a single weather–related Cube time. An illustration in the dynamic Graphic script represented by an interaction diagram, it represents one static framework. Use object diagrams to visualise, define, construct, and record the existence resulting from determined coincidence in his system, as well as those relationships.

❑ **Check Your Progress – 3 :**

1. From the following, object is divided into what parts ?
  - a. Bottom part
  - b. Top part
  - c. Both a & b
  - d. None of these

**9.5 Modelling Techniques for Object Diagrams :**

**Modelling Object structures :**

To model one object structure,

- Decide the mechanism you want to model. A mechanism is a representation of a function or behaviour in the modelled portion of the system that arises through the interaction of a society of classes, interfaces, and other elements.
- Organize a discussion to outline a mechanism.
- For each mechanism, list the classes, interfaces, and other components that are involved; list the connections between these components as well.
- Think about a scenario that utilises this process. At some point, freeze the situation, then duplicate each component of the mechanism.
- As needed to comprehend the scenario, reveal the state and attribute values of each of these objects.
- The links between these objects, which serve as examples of their associations, should also be made clear.

**Forward and Reverse Engineering :**

Although theoretically viable, forward engineering work—the production of code from a model—has minimal practical usefulness. Instances are objects that the application produces and destroys while operating in an object-oriented system. You cannot, then, instance These objects of out precisely.

Using an object diagram for reverse engineering (building a model from code) can be helpful. In reality, you or your tools will frequently perform this task while debugging your system. In order to identify where an item's state or its connections to other objects are broken at any given moment, for instance, you might mentally or physically draw an object diagram of the impacted objects if you are chasing after a dangling link.

To reverse engineer an object diagram,

- Choose the object that you want to reverse engineer. Typically, you will establish your context in connection to an action or a class instance.
- Stop acting at a certain period, either by using a tool or by merely moving around the scene.
- Discover the collection of intriguing objects that interact in that situation and depict them in an object diagram.
- Expose the states of these objects as necessary to comprehend their semantics.
- Identify the relationships that exist between these things if it is important to comprehend their semantics.

**❑ Check Your Progress – 4 :**

1. An attribute is held by \_\_\_\_\_
  - a. Object
  - b. Class
  - c. (a) and (b) both
  - d. None of these
2. An attribute is held by \_\_\_\_\_
  - a. Object
  - b. Class
  - c. (a) and (b) both
  - d. None of these

**9.6 Let Us Sum Up :**

In this unit, we have learned that Class diagrams are one of the main building blocks in object-oriented modelling. We have also learned the terms & concepts of class diagram with how we can use class diagram in different ways for Modelling. Also we have discussed the modelling techniques for class diagram in this unit.

We have also learned that An Object Diagram can be referred to as a screenshot of the instances in a system and the relationship that exists between them. We have also learned the terms & concepts of object diagram with how we can use object diagram in different ways for Modelling. Also, we have discussed the Modelling techniques for object diagram in this unit.

**9.7 Answer to Check Your Progress :**

- ❑ Check Your Progress 1 :**  
1 : d
- ❑ Check Your Progress 2 :**  
1 : d
- ❑ Check Your Progress 3 :**  
1 : c
- ❑ Check Your Progress 4 :**  
1 : c
- ❑ Check Your Progress 5 :**  
1 : c

**9.8 Glossary :**

1. **Class** – they are a group of objects with common properties and are like a model from which the objects are created.
2. **Object** – An object is a diagram that can be referred to as a screenshot of the instances in a system and the relationship that exists between them.

**9.9 Assignment :**

1. Explain class and object in detail.

**9.10 Activities :**

1. Explain Modelling techniques for class diagrams



**9.11 Case Study :**

1. Discuss the Modelling Techniques for Object Diagrams.

**9.12 Further Readings :**

1. Maitri Jhaveri, Madhuri B. Arhunshi – Structured and object-oriented analysis and design methodology –person
2. Norman,Ronald– object oriented system analysis and design –prentice hall 1996

**UNIT STRUCTURE**

- 10.0 Learning Objective
- 10.1 Introduction
- 10.2 Basic Behavioural Modelling : Interactions
- 10.3 Interaction Diagrams
- 10.4 Let Us Sum Up
- 10.5 Answers for Check Your Progress
- 10.6 Glossary
- 10.7 Assignment
- 10.8 Activities
- 10.9 Case Study
- 10.10 Further Readings

**10.0 Learning objectives :**

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

- Study about Interactions
- Study about Creation, Modification, and Destruction
- Study about Modelling the flow of control
- Study about Interaction diagram
- Study about Forward and Reverse engineering

**10.1 Introduction :**

Interaction diagrams are used when we want to understand message flow and structural organization. Message flow means the sequence of control flow from one object to another. Structural organization means the visual organization of elements of a system.

**Interaction diagrams can be used :**

- Model the flow of control by time sequence.
- Model the flow of control from structural organizations.
- For forward and reverse engineering.

**10.2 Basic Behavior Modelling : Interactions :**

**Terms and concepts :**

A collection of messages is transmitted between a group of objects during an encounter in order to accomplish a specific goal. A message is a description of an exchange of information between objects that assumes a subsequent action will be taken.

**Context :**

Anywhere that items are connected to one another, there is an interaction. Collaboration between things that are present in the context of your system or subsystem will reveal interactions. There are interactions connected to operations as well. The interactions related to a class are the last.

The majority of the time, interactions will be found where items that are part of your system or subsystem as a whole work together. For instance, client-side objects that communicate with one another can be found in a web trading system (for example, instances of the BookOrder and OrderForm classes). Additionally, there are client-side objects (again, like instances of BookOrder) that communicate with server-side objects (such as instances of BackOrderManager). Therefore, these relationships can span several conceptual layers of your system as well as localised collaborations of objects (like those around OrderForm) (such as the interactions around BackOrderManager).

The execution of an operation also involves interactions between objects. The algorithm that implements an operation can interact with objects that are global to the operation (but still visible to the operation), local variables for the operation, and parameters for the operation. A mobile robot, for instance, will interact with a parameter (p), a global object of the operation (like the currentPosition object), and maybe more local objects when the moveToPosition operation (p: Position) is activated (such as variable locations, used by the operation to calculate waypoints on a path to the new position).

The interactions related to a class are the last. Interactions can be used to represent, define, build, and document a class's semantics. For instance, you can design interactions that demonstrate how properties for a RayTraceAgent class interact with one another in order to comprehend what that class means (and with objects that are global for class instances and with parameters defined in class operations).

**Objects and roles :**

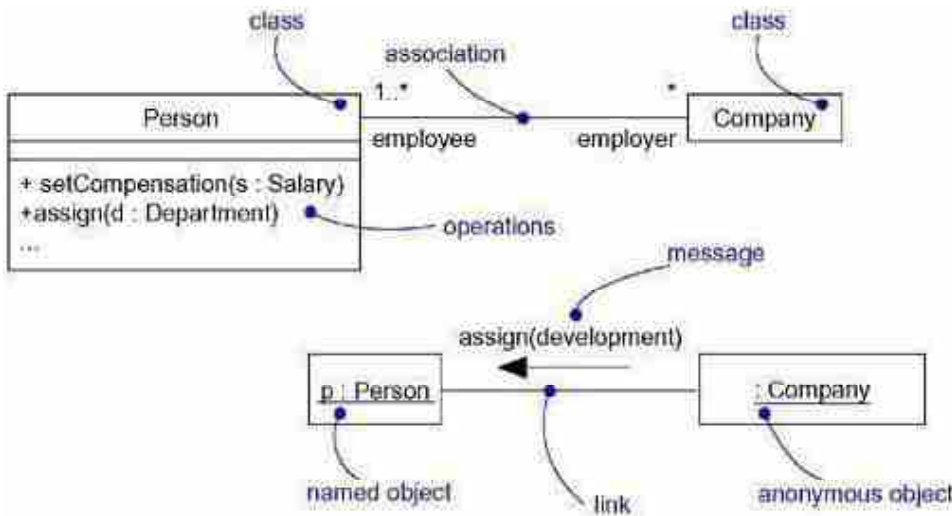
Concrete or archetypal objects make up the participants in an encounter. An object serves as a representation of anything in the physical world. An instance of the class Person, such as p, could be used to represent a specific person. Alternatively, p might stand for any instance of Person as a prototype item.

There are instances of classes, components, notes, and use cases associated with an interaction. Despite the fact that by definition abstract classes and interfaces do not have direct instances, you can nonetheless find instances of these things in a single transaction. Such occurrences may indicate indirect (or prototype) instances, respectively, of any concrete child of the abstract class of a concrete class realising this interface rather than direct occurrences of the abstract class or the interface.

An object diagram can be viewed as a depiction of the static portion of an interaction that identifies all the cooperating items and prepares the scene for the interaction. By adding a dynamic sequence of messages that can be transmitted along the linkages connecting these items, an interaction takes things a step further.

**Links :**

A link connects two objects semantically. A link is typically an example of an association. As shown in the image, instances of two classes may be connected when one class is affiliated with another class. When two objects are connected, one object may send a message to the other object.



**Figure 10.1 : Links**

A link denotes a route that one object can use to communicate with another (or the same) object. Usually, just mentioning the existence of such a route is sufficient. You can add one of the following common stereotypes to the appropriate end of the link if you need to be more specific about how that approach works.

<b>Association</b>	Specifies to that correspondent object is visible by association
<b>Self</b>	Specifies to that correspondent object it is visible because it is that sender of the operation
<b>Global</b>	Specifies to that correspondent object it is visible because it is in one field of application
<b>Local</b>	Specifies to that corresponding object it is visible because it is in one local scope
<b>Parameter</b>	Specifies to that correspondent object it is visible because it is one parameter

**Messages :**

<b>Call</b>	Invoke an operation on an object; one object able to send one message Likewise, results in that local invocation from one operation.
<b>Return</b>	Returns a value to that calls up
<b>Send</b>	Send one sign to one object
<b>Create</b>	Creates one object
<b>Destroy</b>	Destroy an object; an object able to commit suicide to destroying itself

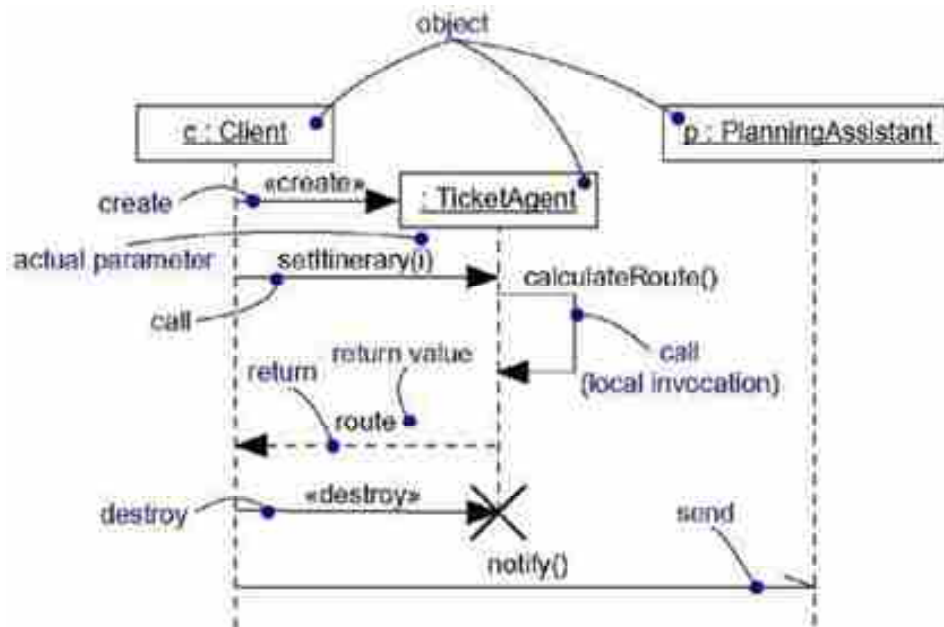
Assume you have a collection of objects and a collection of links connecting them. If that's all you've got, your model is entirely static and can

be portrayed by an object diagram. When you need to depict, specify, develop, or describe a static object structure, object diagrams come in handy since they represent the status of a community of objects at a specific moment.

Let's say you wish to simulate how a group of objects' states change over time. Consider it as capturing a series of objects in motion, with each frame representing a distinct point in time. If these things are not entirely dormant, you will observe them sending events, actions, and messages to other objects. Additionally, you may clearly display the present condition and function of each instance in each box.

The description of a communication between objects that transmits information with the hope that an activity will occur is called a message. One might think of the reception of a message instance as an instance of an event.

An executable statement that represents an abstraction of a calculating technique is the consequence of sending a message. A state can change as a result of an action. You can model several activity kinds in UML. As seen in the picture, the UML offers a visual differentiation between these different types of messages.



**Figure 10.2 : Messages**

The call, where one object calls an operation on another (or the same) object, is the most typical sort of communication you should describe. Any random operation cannot be called by an object. The `setItinerary` action must not only be defined for the `TicketAgent` class (i.e., it must be declared on the `TicketAgent` class or one of its parents) but also be made visible to the caller `c` if an object, like in the example above, calls the action on an instance of the `TicketAgent` class.

An object may include the message's most recent parameters when it calls an operation or sends a signal to another object. You may also model the return value when an object transfers control to another object.

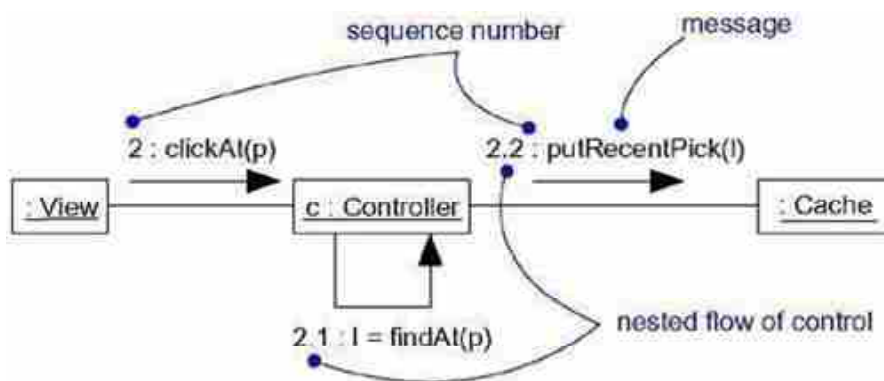
**Sequencing :**

An object can send a message to another object, which effectively delegated an action to the recipient, and the recipient can then send a message to yet another object, and so on. There is a sequence to this message flow.

Each sequence must begin, and each sequence's beginning is anchored in a process or thread. Additionally, any thread will continue for the duration of the process or thread that it belongs to. As long as the node it is operating on is up, an uninterrupted system, like as the one you could see in real-time device control, will keep functioning.

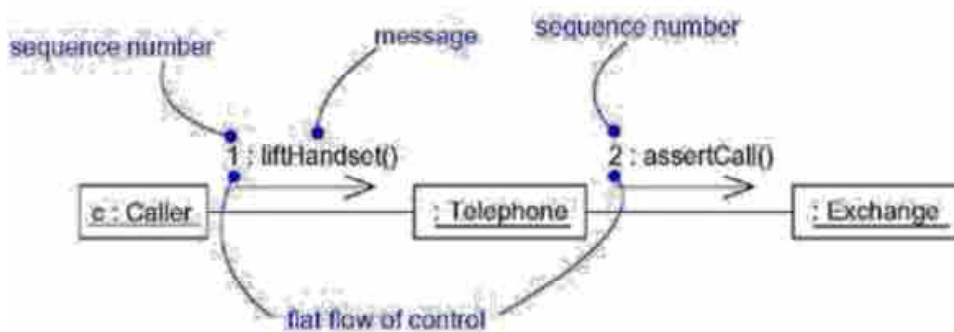
A system's processes and threads define various control flows, and within each flow, messages are ordered according to time. Placing a sequence number in front of the message, separated by a colon, can help you more clearly see how the messages are organised in relation to the beginning of the series.

Typically, you can define a procedural or embedded control stream, which is symbolised in the picture by a filled-in, solid arrowhead. Specifically, the discovery the first message in the sequence that is concealed within the second message is referred to as a message (2.1).



**Figure 10.3 : Sequences**

A flat control flow, denoted by an arrowhead in the picture, can be specified to simulate the less typical but still conceivable evolution of non-procedural control from one step to the next. The assertCall message is designated as the second message in the sequence in this instance.



**Figure 10.4 : Procedural sequence-2**

Finding the process or thread that sent a specific message is especially crucial when modelling interactions involving numerous control streams. By placing the message's sequence number in front of the name of the process or thread that is at the sequence's root, you can tell one control flow from another in UML. For instance, the phrase

**D5:ejectHatch(3)**

specifies that the actual parameter for the ejectHatch action will be transmitted as the fifth message in a sequence that will be anchored by the process or thread D.

You can display the return values for a function in addition to viewing the actual arguments supplied with an operation or a signal connected to an interaction. The p-value from the lookup operation, sent with the current parameter "Rachelle," is returned, as shown by the following expression. This message is part of an embedded sequence that was sent as the second message inside the third message inside the first message. P can be used as a real parameter in other messages in the same diagram.

**Creation, Modification and Destruction :**

The majority of the time, the items you display are taking part in an interaction all the time. However, some interactions allow for the production and destruction of objects (defined by a creation message) (specified by a destruction message). Links are no different in that relationships between items can change. You can provide an item one of the following constraints to determine whether an object or link enters and/or exits during an interaction:

<b>new</b>	specifies to that example either link it is created during performance from that enclosing interaction
<b>destroyed the en</b>	specifies to that example either link it is destroyed Earlier to completion from performance from lose interaction
<b>transient interact</b>	specifies to that example either link it is created during execution from that Unlock ion but it is destroyed before completion from performance

An object often modifies the values of its characteristics, its state, or its functionalities during an interaction. By simulating the object in the interaction, you may depict how an object has changed (possibly with different attribute values, state, or roles). You will place each variation of the object on the same lifeline in a sequence chart. You link each variant to a converted message in an interaction chart.

**Representation :**

Typically, while modelling an interaction, both objects (each with a specified role) and messages are present (each representing the communication between objects with some resulting action).

By emphasising both the structural arrangement of the objects that send and receive messages as well as the temporal order of their messages, you may see the objects and messages engaged in an interaction. The first type of representation in UML is known as a sequence diagram, while the second type is known as a cooperation diagram. Collaboration diagrams and sequence diagrams are both examples of interaction diagrams.

The majority of sequence diagrams and collaboration diagrams are isomorphic, which means you may change one into another without losing information. There are some visual variations, though. Sequence diagrams, in the first place, let you model an object's lifeline. The existence of a thing is represented by its lifeline, which may also include its creation and demise. The second benefit of collaboration diagrams is that they let you model any structural links that could exist between the objects involved in an interaction.

**Common Modelling techniques :**

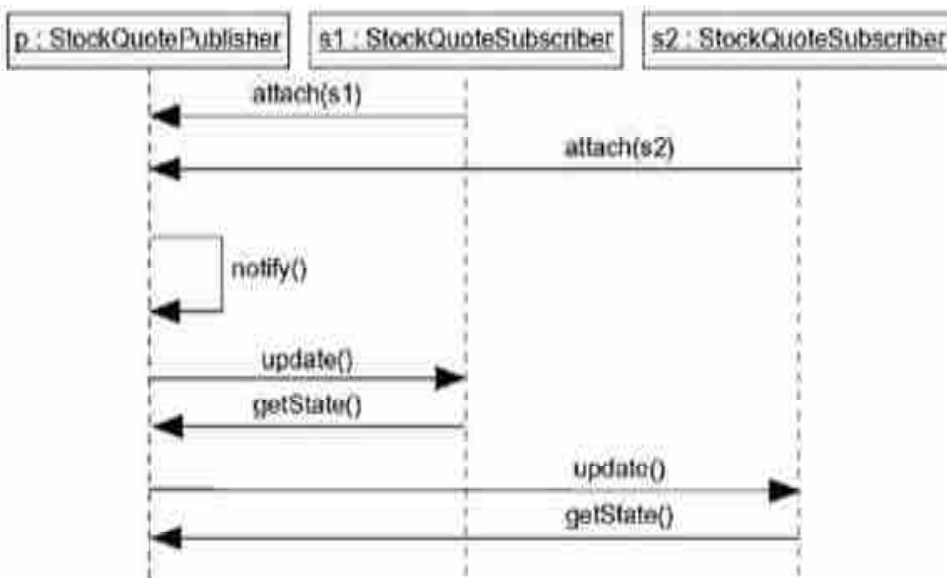
**Modelling a Flow of Control :**

A storyboard of the interactions between a group of objects is essentially what you are making when you model an interaction. It is very helpful to employ methods like CRC cards to find and consider these interactions.

To model a flow of control,

- Set the context for the interaction, whether it involves the system as a whole, a class, or a distinct operation.
- Establish the initial properties of the objects, such as their attribute values, states, and roles, to set the stage for the interaction.
- If your model focuses on the structural arrangement of these things, note the connections between them that are pertinent to the communication channels used in this interaction. Use common UML stereotypes and restrictions, such as needed, to describe the types of linkages.
- Indicate the sequence in which messages should be passed from one object to another. As needed, differentiate between the various message types; provide parameters and return values to pass the necessary information from East–West interaction.
- Along with providing the information from the East interaction, embellish every object at all times with his role and state.

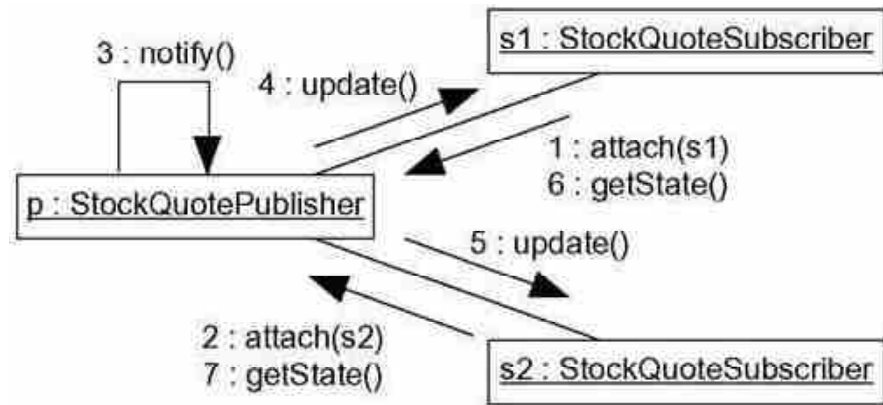
The illustration, for instance, depicts a group of interconnected items as part of a publishing and subscription process (an instance of the observer design pattern). Three objects are shown in this figure: p (aStockQuotePublisher), s1, and s2 (both instances of StockQuoteSubscriber). This diagram illustrates a sequence chart that places emphasis on the messages' chronological order.



**Figure 10.5 : Sequence Chart**

Despite being designed as a collaborative diagram that emphasises the structural structure of the objects, Figure is semantically comparable to the preceding one. The links between these objects are also visualised in this picture, which not only depicts the same control flow.





**Figure 10.6 : Sequence Chart**

**□ Check Your Progress – 1 :**

1. Which of the following are used to model dynamic aspects of collaborations.
  - a. Interaction
  - b. Structural
  - c. Sequence Diagrams
  - d. Messages
2. There are \_\_\_\_\_ types of interaction diagrams.
  - a. 4
  - b. 2
  - c. 6
  - d. 3
3. \_\_\_\_\_ does not include in message types.
  - a. return
  - b. call
  - c. delete
  - d. send

**10.3 Interaction Diagrams :**

**Terms and Concepts :**

An interaction diagram depicts an interaction made up of a group of objects, their connections, and any messages that may be exchanged between them. An interaction diagram that places emphasis on the timing of messages is called a sequence diagram. A sequence diagram is a table with objects plotted along the X-axis and messages plotted along the Y-axis in ascending time order. The structural structure of communications and the objects that send and receive messages are highlighted in collaboration diagrams, which are interaction diagrams. A cooperation diagram looks like a collection of arcs and corners.

**Common features :**

A name and graphic content that is a projection on a model are two qualities that all diagrams have in common, making interaction diagrams just a special kind of diagram. An interaction chart's unique content is what distinguishes it from all other chart kinds.

**Contents :**

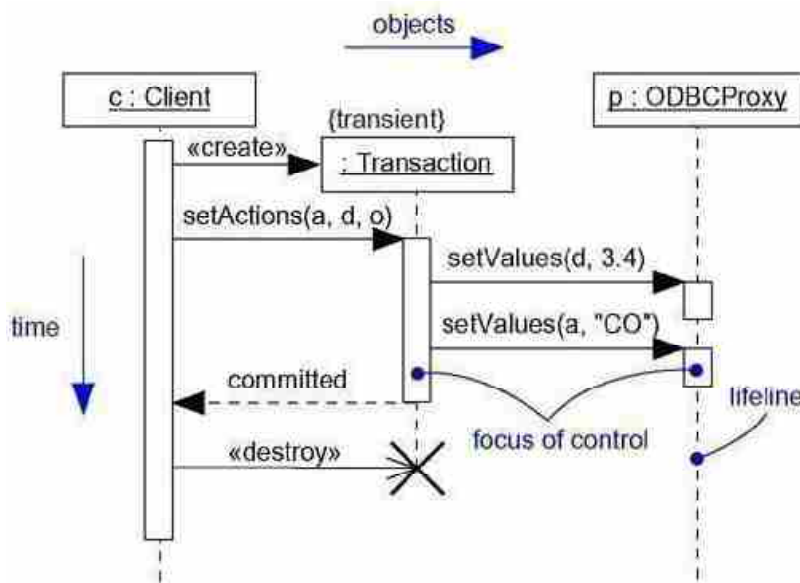
Interaction diagrams Generally Contains

- Objects
- Links
- Messages

As like all other diagrams, interaction diagrams may contain constraints and notes.

**Sequence Diagrams :**

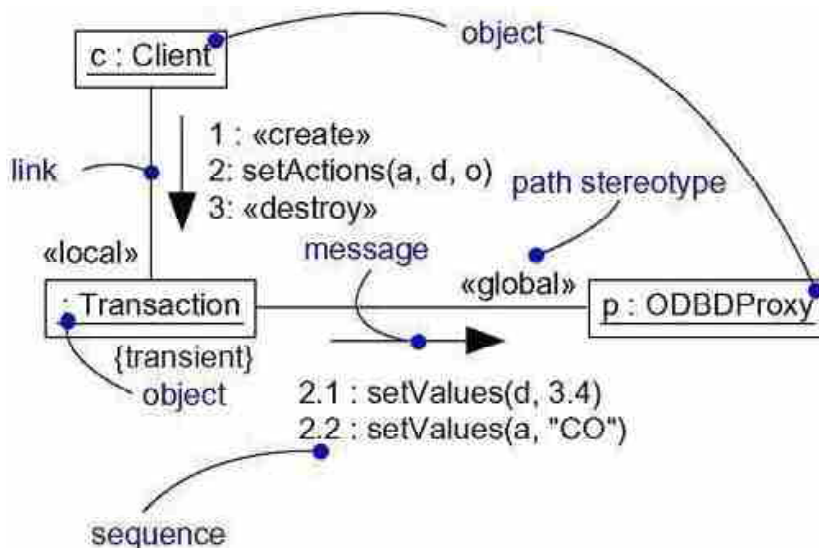
A sequence chart highlights the communications' temporal order. The items involved in the interaction are initially positioned at the top of your design along the X-axis to create a sequence diagram, as shown in the picture. The object that initiates the interaction is typically placed on the left, with progressively more child-friendly toys placed on the right. After that, arrange the messages that these objects transmit and receive along the Y-axis in ascending time order. This provides the reader with a crystal-clear visual representation of the control flow across time.



**Figure 10.7 : Sequence Diagram**

**Collaboration diagrams :**

A collaboration diagram places emphasis on how the objects involved in an interaction are organised. The items involved in the interaction are initially positioned as corners on a graph, as shown in the picture, to create a cooperation diagram. The arcs in this graph should then be used to illustrate the ties connecting these things. The messages that the objects transmit and receive are then used to embellish these linkages. This provides the reader with a crystal-clear visual cue regarding the direction of control within the framework of the cooperating items' structural organisation.



**Figure 10.8 : Collaboration diagrams**

Sequence diagrams and collaboration diagrams serve two different purposes.

The road is the first thing to mention. You can add a route stereotype (such "local," which denotes that the specified object is local to the sender) to the other end of a link to designate how one object is related to another. In general, only the local, parameter, global, and custom (but not association) pathways need to explicitly express the binding path.

The sequence number is the second. A number should be placed in front of each message in the control flow to indicate the message's time order, starting with message number 1 and monotonically increasing for subsequent messages (2, 3, etc.). Use Dewey decimal numbering to display embedding (1 is the first message, 1.1 is the first message embedded in message 1, 1.2 is the second message embedded in message 1, and so on). Viewing embedding can be done at any depth. Remember that multiple messages (potentially delivered from other addresses) can be displayed along the same link, and each will have a separate sequence number.

You will mostly model direct and sequential control flows. It is possible to represent more intricate flows that iterate and branch, though. An iteration is a recurring series of communications. Prefixing a message's sequence number with an iteration expression, such as \* I = 1..n] (or just \* if you want to define the iteration but not its specifics) will simulate an iteration.

An iteration signifies that the message will iterate in accordance with the provided expression, as well as any embedded messages. In a similar manner, a condition denotes a message whose execution is contingent on the outcome of a Boolean condition. A conditional statement, such as [x > 0], should come before a message's sequence number in order to depict a condition. The sequence numbers of alternate paths on a branch will be the same, but each path must be clearly distinct with a non-overlap requirement.

UML does not specify the format of the expression in parentheses for iteration or branching; you are free to use pseudocode or a particular programming language's syntax.

#### **Semantic Equivalence :**

Both sequence diagrams and collaboration diagrams draw from the same data in the UML meta-model, making them semantically similar. As you can see in the two semantically comparable figures above, you can transfer a chart from one form to another without losing any information. This does not imply that both charts directly display the same data, though.

For instance, the matching sequence diagram does not depict the connections between the objects as the cooperation diagram does (notice the stereotypes "local and" global). Similar to how the split chart indicates the return of the message but the sequence chart does not (notice the needed return value). The two diagrams in each case have the same underlying model, but each one might depict some things that the other might not.

#### **Common Uses :**

It models a system's dynamic elements using interaction diagrams. Any type of instance from any view of the architecture of a system may interact in these dynamic aspects, including instances of classes (including active classes), interfaces, components, and nodes.

When modelling a dynamic aspect of a system with an interaction diagram, you can do it within the context of the system as a whole, a subsystem, an operation, or a class. To represent a scenario, you may also add interaction diagrams to use cases and collaborations (to model the dynamic aspects of a community of objects).

Interaction diagrams are commonly used in two ways when modelling a system's dynamic components.

**1. To model flows of control by time ordering**

Here, sequence diagrams are required. The emphasis on message transmission as it changes over time in a timed control flow model makes it a particularly helpful tool for visualising dynamic behaviour in the context of use-case scenarios. Collaboration diagrams are less effective at depicting straightforward iterations and branches than sequence diagrams.

**2. To model flows of control by organization**

Use collaborative diagrams in this situation. When an organisation models a control flow, it stresses the structural connections that exist between the participating bodies and along which information can travel. In comparison to sequence diagrams, collaboration diagrams are more effective in representing intricate iterations and branches as well as more concurrent control streams.

**Common Modelling techniques :**

**Modelling Flows of Control by Time Ordering :**

Think about the items that make up a system, subsystem, operation, or class. A collaboration or use case should also take into account the objects and roles involved. Use a sequence diagram, a type of interaction diagram, to highlight the passage of messages as they change over time. Use an interaction diagram to depict a control flow that passes through these items and roles.

To model a control flow by time order,

- Establish the interaction's context, specifying whether it pertains to a system, subsystem, operation, class, one use case scene, or collaboration.
- Establish the context for the encounter by identifying the objects that will be involved. The most significant things should be placed to the left of his neighbouring objects in the sequence chart, from left to right. a right angle.
- Set a lifeline for every item. The objects will typically stay in place throughout the encounter. Set lifelines for items that are generated and destroyed during interaction, such as suitable posts that explicitly depict Y's birth and death.
- Design each following message from the top down between the lifelines that illustrate the properties of each message (such as its parameters), as necessary to explain that semantics from the interaction. Begin with the message that initiates this interaction.
- Decorate every object Lifeline with his focus from control if you need to visualise that embedding from posts or that point in the weather when current computation is taking place.

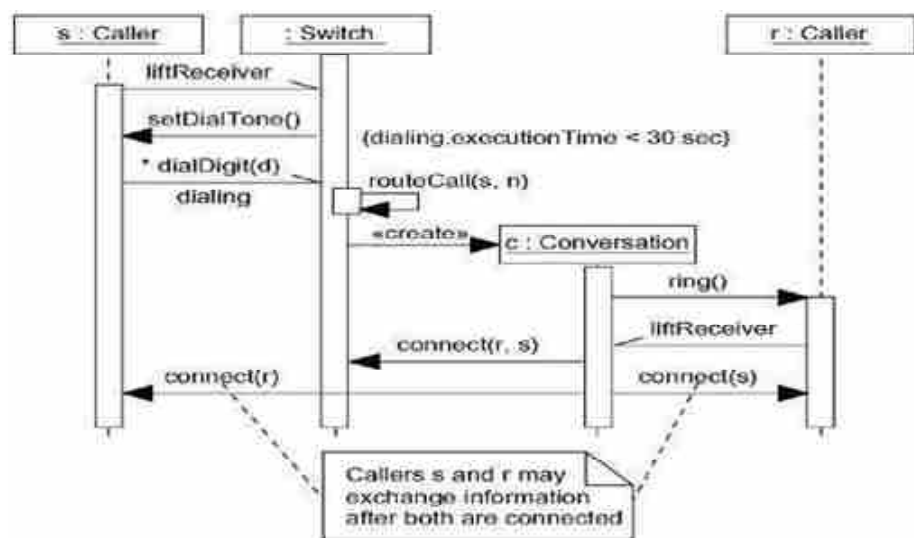
**Object Oriented  
Analysis and Design**

- If time or space restrictions need to be specified, decorate each message with a synchronisation framework Y and attach the necessary deadlines or restrictions.
- If you need to formally identify East flow from control plus, add the following postal conditions to each message: before-Y.

Only one control flow can be displayed in a single sequence diagram (although it can show simple variations using the UML notation for iteration and branching). A succession of interaction graphs, some of which are primary and others which represent alternative paths or remarkable interactions, will typically be present. These sets of sequence diagrams can be organised using packages, and each diagram can be given a unique name to set it apart from the others.

The picture, for instance, displays a sequence diagram outlining the control flow for starting a straightforward phone call between two people. Four objects are present at this level of abstraction: two callers (syr), an anonymous phone contact, and c, which represents the realisation of the two parties' dialogue. A caller initiates the sequence by providing the Switch object a signal (liftReceiver). The switch then instructs the caller to dial setDialTone, and the caller then repeats the dialCifer message. The time stamp (markup) on this message serves as a time restriction (its execution time must be less than 30 seconds).

What happens if this deadline is missed is not shown in this graphic. You can do it by including a totally different branch or sequence diagram. The routeCall message is then sent by the switch object to itself. The remaining task is then transferred to the conversation object (c), which was just created. Despite not being depicted in this exchange, c would also be in charge of the switch's billing system (which would be expressed in another interaction diagram). The liftReceiver message is sent asynchronously by the call object (c), which then calls the caller (s). After asking both callers to connect, the caller requests the contact to connect the call so that they can share information as stated in the following note.



**Figure 10.9 : Modeling Flows of Control by Time Ordering**

At any point in a series, an interaction diagram can begin or finish. The division of a big flow into smaller flows makes sense because a complete control flow track would be quite complicated.

**Modelling of control flows by organization :**

Think about the items that make up a system, subsystem, operation, or class. A collaboration or use case should also take into account the objects and roles involved. Use an interaction diagram to represent the control flow that flows between these objects and roles; a collaboration diagram, a type of interaction diagram, is needed to depict the message that is passed within the context of that structure.

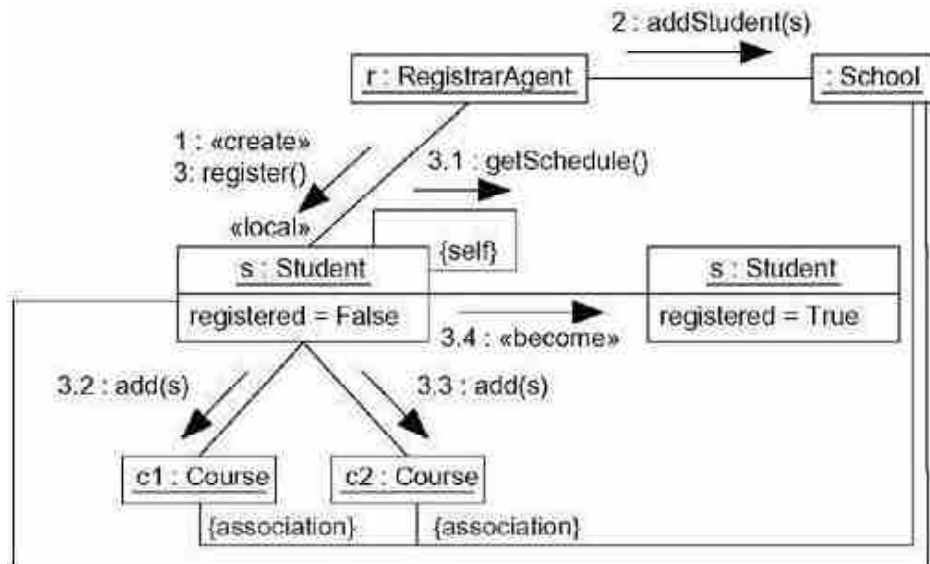
To model a flow of control by organization,

- Establish the interaction's context, specifying whether it pertains to a system, subsystem, operation, class, one use case scene, or collaboration.
- Establish the context for the encounter by identifying the objects that will be involved. Place the most significant elements in the centre of the cooperation diagram and neighbouring things to the sides, like the corners of a graph.
- Set each of these items' initial characteristics. Place a duplicate of the object in the chart, update it with the new data, and connect them using the message stereotypes become or copy if any item's attribute values, tagged values, status, or role significantly changes throughout the interaction (med one appropriate sequence number).

A single collaboration diagram can only display one control flow, much like with sequence diagrams (although it can show simple variations using the UML notation for interaction and branching). There will typically be a number of these interaction diagrams, some of which will be primary and others of which will represent alternate paths or remarkable links. These collections of collaborative diagrams can be organised using packages, and each diagram can be given a unique name to set it apart from the others.

In the picture, for instance, a cooperation diagram detailing the control flow associated with enrolling a new student in a school is shown, with a focus on the structural connections between these elements. A student (s), two course objects (c1 and c2), a RegistrarAgent (s), and an unnamed school object are the five objects you will see. The numbering of the control flow is explicit.

The RegistrarAgent first creates a student object, adds the student to the institution (using the addStudent message), and then requests that the Student object register. The student object then uses the getScheduleon method on itself to obtain the course objects it requires for registration. Each course object is then given the student object. The stream concludes with the letter s rendered once more, indicating that its registered attribute now has a new value.



**Figure 10.10 : Modeling of control flow by organization**

Even though no messages are displayed along these channels, this figure indicates a link between the school object and each of the two course objects, as well as another link between the school object and the student object. These links provide more information on how the student object can view the two course objects to which it has been added, indirectly through the educational object (collection of course objects).

**Forward and reverse engineering :**

Both sequence and collaboration diagrams allow for forward engineering (the production of code from a model), particularly when the context of the diagram is an operation. For instance, a moderately clever advanced engineering tool may produce the following Java code for the operation log connected to the Student class using the collaboration model shown above.

```
public void register() {
    CourseCollection cc = getSchedule();
    for (int j = 0; j<c.size(); j++)
        cc.item(j).add(this);
    this.registered = true;
}
```

If a tool is "quite intelligent," it should be able to deduce from looking at the operation's signature that getSchedule returns a CourseCollection object. The code can then be applied to any number of course offerings by going over the contents of this object using a standard iteration language (which the tool may already be familiar with).

Both sequence and collaboration diagrams can be reverse engineered (to build a model from code), especially if the context of the code is the body of an operation. The segments in the aforementioned diagram might have been created using a tool from a prototypical registration activity.

**❑ Check Your Progress – 2 :**

1. Which diagram from the following demonstrate the interaction which consists a set of objects, its relationship with messages.
  - a. class
  - b. object
  - c. activity
  - d. interaction

2. \_\_\_\_\_ engineering of a collaboration diagram is possible If the context of the diagram is an operation
- a. forward
  - b. backward
  - c. both (a) and (b)
  - d. None of these

#### **10.4 Let Us Sum Up :**

In this unit, we have learned that how Interaction diagrams are used when we want to understand message flow and structural organization. Also discussed the Message flow means the sequence of control flow from one object to another. We have also seen that Structural organization means the visual organization of elements of a system.

We also discussed the sequence diagram and collaboration diagrams in detail. Also we had discussed the modelling the flow of control. At last you also learnt the forward engineering and reverse engineering with example.

#### **10.5 Answers to Check Your Progress :**

**Check Your Progress 1 :**

1 : a          2 : b          3 : c

**Check Your Progress 2 :**

1 : d          2 : a

#### **10.6 Glossary :**

- 1. **Link** – A link is a semantic connection between objects.
- 2. **Interaction** – A collection of messages are transmitted between a group of objects during an encounter in order to accomplish a specific goal.

#### **10.7 Assignment :**

- 1. What is the purpose of behavior Modelling and when should it be used ?

#### **10.8 Activities :**

- 1. Explain sequence diagram in detail.

#### **10.9 Case Study :**

- 1. Discuss Forward and reverse engineering.

#### **10.10 Further Readings :**

- 1. Maitri Jhaveri, Madhuri B. Arhunshi – Structured and object-oriented analysis and design methodology – persona
- 2. Norman, Ronald – Object Oriented Systems Analysis and Design – Apprenticeship Hall 1996



**BLOCK SUMMARY :**

In this block, you learned the unified Modelling language is and what is the importance of Modelling for application development and object-oriented Modelling with two different approaches to developing any model. You also learnt the difference types of relationship in Modelling, the advanced classes and advanced relationship for Modelling. You also learn the interfaces, types of roles and packages with diagram in Modelling.

In this block, you also learned that Class and object diagrams are one of the main building blocks in object-oriented Modelling. You also leaned the terms & concepts of class and object diagram with how we can use class and object diagram in different ways for Modelling.

In this block you also learnt the sequence diagram and collaboration diagrams in detail. Also we have discussed the Modelling the flow of control. At last you also learnt the forward engineering and reverse engineering with example in this block.

## **BLOCK ASSIGNMENT :**

### **❖ Short Questions :**

1. What do you understand by UML ?
2. What is relationship in UML ?
3. What is static diagram in UML ?
4. What are the dynamic diagram & messages in UML ?

### **❖ Long Questions :**

1. Explain Interaction diagram with diagram.
2. Explain class and object diagram in detail.
3. How many diagrams are there in UML ? Explain any one diagram other than mentioned in question-1 and 2.

**Object Oriented  
Analysis and Design**

❖ **Enrolment No. :**

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

Unit No.	7	8	9	10
No. of Hrs.				

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

Items	Excellent	Very Good	Good	Poor	Give specific example if any
Presentation Quality	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Language and Style	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Illustration used (Diagram, tables etc)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Conceptual Clarity	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Check your progress Quest	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Feed back to CYP Question	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____

3. Any other Comments

.....

.....

.....

.....

.....

.....

.....

.....



**Dr. Babasaheb Ambedkar  
Open University Ahmedabad**

**BCAR-502**

# **OBJECT ORIENTED ANALYSIS AND DESIGN**

---

## **BLOCK 4 : UNIFIED MODELLING LANGUAGE-II**

---

UNIT 11 BASIC BEHAVIORAL MODELING-II

UNIT 12 ADVANCED BEHAVIORAL MODELING

UNIT 13 ARCHITECTURAL MODELING

UNIT 14 CASE STUDY

# ***UNIFIED MODELLING LANGUAGE-II***

## **Block Introduction :**

Unified Modelling (UML) a general graphic modeling language in software engineering. UML is used to specify, view, create and document artifacts in the software system. It was originally developed by Grady Booch, Ivar Jacobson and James Rumbaugh in 1994–1995 at Rational Software, where development continued until 1996. In 1997, the Object Management Group adopted it as the standard.

In this block we will describe in detail the meaning of behavioural modelling and which diagrams we can make in it. Also, we will describe the use case diagram and activity diagram in detail. We will also describe the advanced behavioural modelling with detail about events and signal, state machine, processes and threads, Time and space and state chart diagrams.

We will also learn Architectural modelling with detail about component, deployment, and component and deployment diagram in detail. At last we will also learn two case studies about library management system and online mobile recharge. We will discuss all UML diagrams for these two systems.

## **Block Objectives :**

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

- About the behavioural modelling
- About the Use case diagram
- About the Activity diagram
- About Advanced behavioural modelling
- About state chart diagrams
- About architectural modelling
- About component diagram
- About deployment diagram
- About case study of two system

**Block Structure :**

**Unit 11 : Basic Behavioural Modelling–II**

**Unit 12 : Advanced Behavioural Modelling**

**Unit 13 : Architectural Modelling**

**Unit 14 : Case Study**

**UNIT STRUCTURE**

- 11.0 Learning Objective
- 11.1 Introduction
- 11.2 Basic Behavioural Modelling : Use Case
- 11.3 Use Case Diagrams
- 11.4 Activity Diagrams
- 11.5 Let Us Sum Up
- 11.6 Answers for Check Your Progress
- 11.7 Glossary
- 11.8 Assignment
- 11.9 Activities
- 11.10 Case Study
- 11.11 Further Readings

**11.0 Learning Objectives :**

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

- The basics of behavioural modelling
- About the use case and use case diagrams
- About the Activity and activity diagrams

**11.1 Introduction :**

Behavioral Modeling is an approach used by companies to better understand and predict consumer actions. Behavioral modeling uses available data on consumer and business spending to estimate future behavior in specific circumstances.

The original purpose of the use cases was to express user requirements in a technology independent way using a structured narrative form that both the non-IT user and the business analyst can understand. They have also provided a simple yet powerful structure that provides valuable information about the users of the system and how they are using it.

Activity charts show the order of activities in a process, including sequential and parallel activities, and the decisions made. An activity chart is usually created for a use case and can show different possible scenarios.

**11.2 Basic Behavior Modeling : Use Case :**

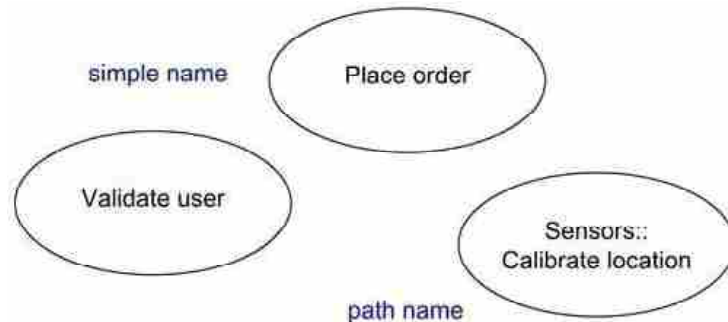
**Concepts and Concepts :**

A use case is a description of a collection of action sequences, including variations, that a system employs to generate a valuable observable result for an actor. A use case is graphically shown as an ellipse.



**Names :**

Each use case needs to be given a name that sets it apart from others. A name is a string of text. A path name is the name of the use case followed by the name of the package the use case is in; this name is only known as a simple name. Typically, a use case is illustrated with just its name visible, like in the illustration.



**Figure 11.1 : Simple and Path Names**

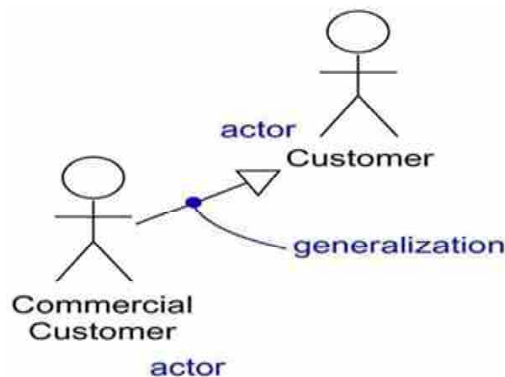
**Note :**

A use case name can span numerous lines of text and contain any combination of letters, numbers, and punctuation—with the exception of those needed to demarcate class and package names, such as the colon. Use case names are essentially condensed active verb phrases that identify certain actions seen in the system's lexicon that you are modelling.

**Use cases and actors**

When use case users interact with these use cases, they take on a cogent set of roles that are represented by an actor. A typical actor portrays a role that a person, a piece of hardware, or even another system might play with a system. You could be a loan manager, for instance, if you work for a bank. You will also take on the role of the Client if you do your personal banking there. Therefore, an actor instance represents a person who engages with the system in a specific way. The actors aren't actually a part of the system, even if you want to incorporate them in your models. They are not part of the system.

The actors are depicted as stick figures, as the figure shows. Using generalisation relationships, you can define general classes of actors (like customers) and specialise them (like commercial customers).



**Figure 11.2 : Actors**

Actors and use cases can only be connected through association. When an actor and a use case are associated, it signifies that the actor and the use case are in communication with one another and exchange messages.

**Use cases and flow of events :**

Use cases give a system's (or subsystem's, class's, or interface's) function without elaborating on how it accomplishes it. Maintaining the worry-free barrier between this on the outside and this within is crucial while modelling.

A text flow of events that can be easily understood by a total stranger can be used to specify the behaviour of a use case. You must describe how and when the use case starts and finishes, when it interacts with the actors and what objects are traded, as well as the fundamental flow and alternative flows of behaviour, while writing this event flow.

You can, for instance, express the ValidateUser use case in relation to an ATM system as follows :

**Main flow of events :**

The use case starts when the system requests a PIN code from the user. A PIN code can now be entered by the consumer via the keypad. The user clicks the Enter button to confirm their entry. The system then verifies the validity of this PIN. The system validates the entry and ends the use case if the PIN code is legitimate.

**Exceptional flow of events :**

- ✓ By clicking the Cancel button, a client can halt a transaction at any point, restarting the use case. The Client's account is left unchanged.
- ✓ Remove a PIN at any moment before to confirmation and before entering a new PIN.
- ✓ The use case will continue if the customer enters an invalid PIN code. The technology cancels the entire transaction if it occurs three times in a row, stopping the consumer from using the ATM for 60 seconds.

**Use cases and scenarios :**

The typical order of events for an in-text use case is to be described first. However, as you gain a better grasp of system requirements, you'll also start using interaction diagrams to visually express these flows. A sequence diagram is typically used to show the primary flow in a use case and other versions of that diagram to show the unusual flows in a use case.

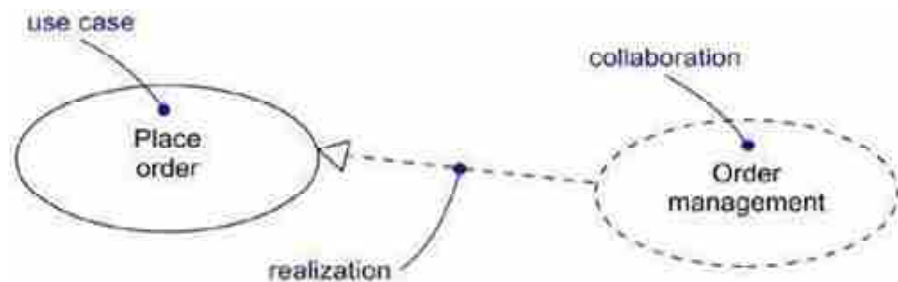
Because a use case describes a collection of sequences rather than just one, it is preferable to divide the primary and alternate streams because it would be impossible to fully communicate the nuances of an interesting use case in a single sequence., and a,,, a gloomy, and a gloomy, There are numerous potential modifications for this general business function. You can move a worker from one department to another (common in global firms), hire someone from another company, or hire a foreign national, which is the most frequent option (which has its own special rules). These variations can all be expressed in various orders.

In reality, this use case presupposes a collection of sequences, where each sequence, taken as a whole, indicates a potential path through all of these variations. Each series is referred to as a scenario. A scenario is a predetermined series of events that depicts behaviour. A scenario is essentially an instance of a use case because scenarios employ use cases as instances for classes.

**Use cases and collaborations :**

Without having to define how that behaviour is implemented, a use case captures the anticipated behaviour of the system (or subsystem, class, or interface) it is building. This is a crucial distinction because, to the greatest extent possible, implementation concerns should not influence the analysis of a system, which describes the behaviour (which specify how that behaviour will be performed). However, you must ultimately put your use cases into action. To achieve this, you must assemble a group of classes and other components that cooperate to carry out the use case's behaviours. This group of elements is modelled in UML as a collaboration, together with their static and dynamic structure.

You can expressly state whether a cooperation is creating a use case, as seen in the figure. However, in the majority of cases, a certain use case is precisely achieved by collaboration, thus you are not required to formally model this relationship.



**Figure 11.3 : Use cases and collaborations**

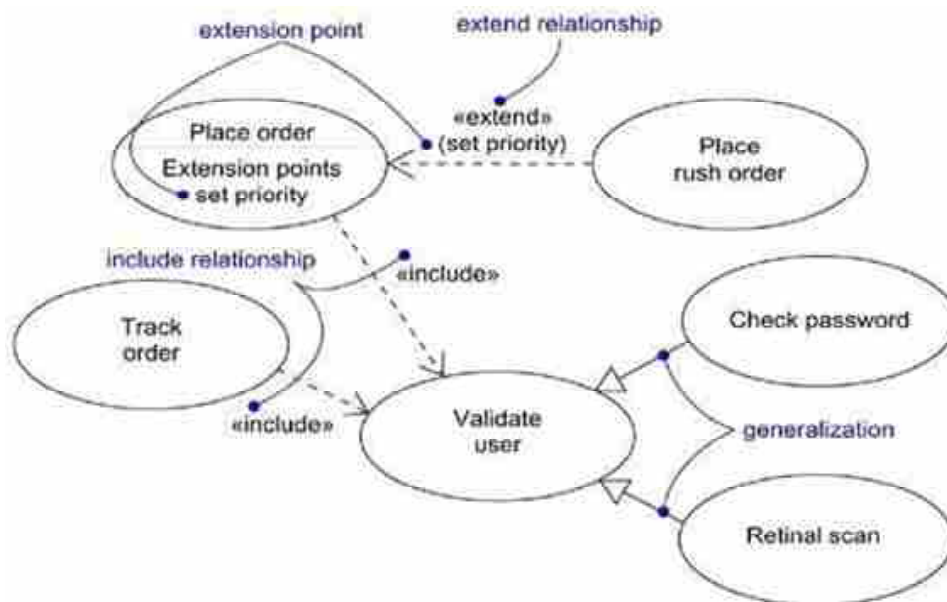
**Organization Use cases :**

Similar to how classes are organised, use cases can be organised by placing them in groups called packages.

By defining the generalisation, inclusion, and extension relationships that exist between the use cases, you can further arrange them. These connections are used to factor variations and common behaviour (by deducting the behaviour from other use cases it comprises) (pushing that behaviour to other use cases that extend it).

The same principles apply to generalising across use cases as they do between classes. In this context, it means that the conduct and significance of the parental case are carried over into the child abuse case; the child may add or delete the father's behaviour; and the child may be substituted wherever the parent appears (both parent and child can have specific cases). For instance, the User Validation case in a financial system is in charge of confirming the user's identification.

Then, you can have two specialised children of this use case (Password Check and Retinal Scan), both of which function as User Validation and can be used anywhere User Validation appears but each of which adds its own behaviour (the first when checking a password of text, the latter when checking the user's distinctive retinal patterns). Similar to generalisation across classes, generalisation between use cases is depicted in the picture as a straight solid line with a broad open arrowhead.



**Figure 11.4 : Generalization, include and extend**

A use case inclusion relationship occurs when one use case at a certain place in the base expressly integrates the behaviour of another use case. The included use case is only ever instantiated as a part of a bigger base that also includes it; it is never used independently. Include can be viewed as a base use case that pulls the desired behaviour from the provider's use case.

By including typical behaviour in your use case, you can avoid explaining the same event flow more than once (the use case included in a base use case). Delegation is mostly demonstrated by the connection of inclusion. You gather all of the system's obligations into one location (using a use case), then permit additional use cases to use the new accountability bundle as necessary. such capability.

It portrays a connection of inclusion as an addition, stereotypically as it encompasses. Simply type include followed by the name of the use case you want to include, as in the following order tracking flow, to describe the point in a sequence of events where the base use case includes the behaviour of another.

**Common modelling techniques :**

**Modeling the behavior of an element :**

Modelling the behaviour of an element, characteristics of the system as a whole, characteristics of a subsystem, or characteristics of a class are the most frequent uses of use cases. It's crucial to concentrate on what the element does, rather than how it does it, while modelling the behaviour of these objects. Applying use cases in this manner to things is crucial for three reasons.

First, it enables domain experts to sufficiently specify an element's external view so that developers can create their own internal view by modelling the behaviour of an element using use cases. Domain experts, end users, and developers can interact with each other through use cases.

Second, use cases give programmers a means to approach and comprehend an element. A system, subsystem, or class might be intricate and packed with many actions and components. You can direct users of those elements based on how likely they are to use them by defining use cases for each element.

## Object Oriented Analysis and Design

Users must learn how to use these elements on their own in the absence of such use cases. Use cases enable the author of an element to express their intention regarding the intended usage of that element.

Third, use cases provide a framework for testing each component as it changes throughout the development process. You continuously validate your implementation by testing each part against its use cases. In addition to serving as a source for regression testing, these use cases compel you to re-evaluate your implementation for each new use case you add to an element in order to make sure it is adaptable. If not, make the appropriate architecture corrections.

To model the behavior of an element,

- Determine who is acting in the interaction with the element. Groups that require specified conduct to carry out their obligations or that are directly or indirectly required to carry out the functions of the element are considered candidate actors.
- Assign roles to the players, both generic and more specific.
- Take into account the main ways that each actor interacts with the object. Additionally, interactions that alter the state of the element or its surroundings or entail a response to an event are taken into account.
- Also take into account the many ways that each actor engages with the object.
- Group these behavioural patterns into use cases, quantify common behaviours, and identify exceptional behaviours using inclusion and extension relationships.

A retail system, for instance, will communicate with customers who are tracking their orders. The system will then send orders and charge the buyer in return. By designating these behaviour models as use cases, as shown in the image, you may simulate the behaviour of such a system (placing an order, order tracking, shipping order, and customer invoices). Normal behaviour can be identified from deviations and excluded (customer validation) (Send Suborders). You must provide the behaviour for each of these uses, whether it is text, a state machine, or an interaction.

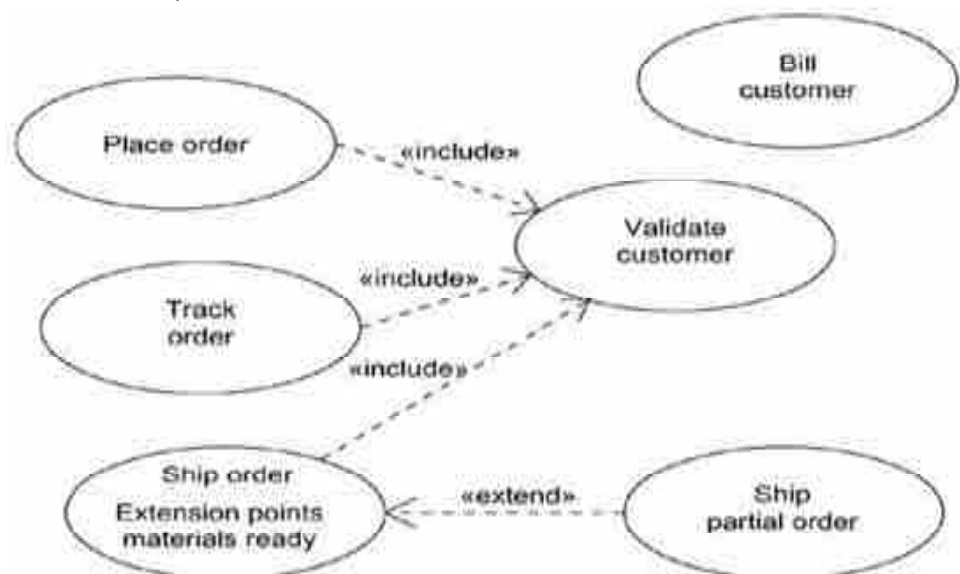


Figure 11.5 : modelling the behaviour of an element

□ **Check Your Progress – 1 :**

1. \_\_\_\_\_ scenario is a specific sequence of action  
a. Use case      b. Class      c. Activity      d. Object
2. \_\_\_\_\_ shape is used within the rectangle to represent use cases.  
a. circle      b. ellipses      c. line      d. rectangle

**11.3 Use Case Diagrams :**

**Terms and concepts :**

A use case diagram is a visual representation of a group of actors, use cases, and their relationships.

**Common features :**

A use case diagram is only a particular kind of diagram, and all diagrams have the same basic elements: a name and graphic content that is projected onto a model. A use case diagram's unique content is what distinguishes it from all other graph types.

**Contents :**

Use case diagrams Generally to contain

- Use cases
- Actors
- Dependence, generalization, and Association relations

Use case charts, like other charts, can have annotations and restrictions.

Packages are also a component of use case diagrams and are used to organise model components into larger blocks. On occasion, you'll also want to include use case examples on your charts, particularly if you want to illustrate a particular execution system.

**Common uses :**

Use case diagrams are used to represent a system's static use case view. This perspective mainly supports how a system behaves and the externally visible services it offers in the context of its surroundings.

Use case diagrams can be used in two ways when representing the static view of a system's use cases.

**1. To model the context of a system**

When modelling the context of a system, the entire system is encircled, and the players who interact with the system from outside are identified. Use case diagrams are used here to identify the actors and the significance of their roles.

**2. To model the requirements for a system.**

Regardless of the method the system should use, modelling the needs of a system entails defining what the system should perform (from an outside perspective to the system). Use case diagrams will be used in this instance to specify the required system behaviour. In this approach, a use case diagram enables you to view the entire system as a black box; you can observe what is occurring outside the system and how it responds to it, but you are unable to understand how the system functions internally.

➤ **Common modeling techniques :**

**Modeling of context of a system :**

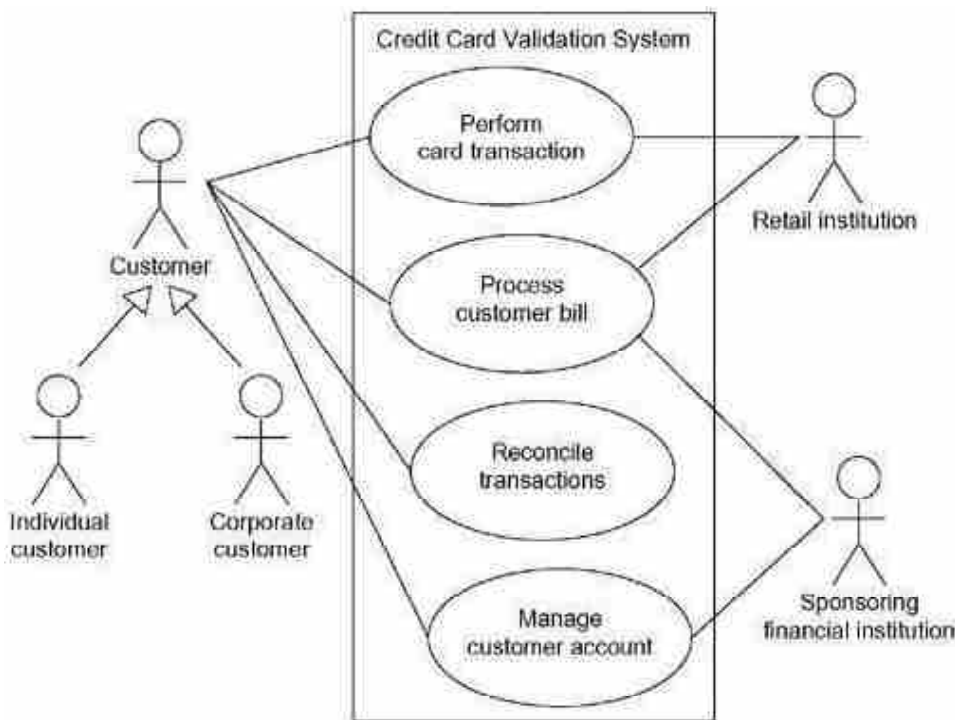
Given a system any system some things will live within the system, some things will live outside of it. In a credit card validation system, you will find eg. things like accounts, transactions and agents to detect fraud in the system. Likewise, you will find things like credit card customers and retail institutions outside the system. Things that live within the system are responsible for performing the behaviours that external ones expect the system to provide. All external things that interact with the system constitute the context of the system. This context defines the environment in which the system lives.

A use case diagram in UML that emphasises the players surrounding the system can be used to model the context of a system. Making a decision about what to include as an actor is crucial since it identifies a group of objects that can interact with the system. Making choices on what not to include as an actor is equally, if not more, crucial since it restricts the system environment to only those players that are required for the system to function.

To model the context of a system,

- Define the actors that surround the system, taking into account the groups that use the system to help them with their tasks, the groups that are required to carry out system functions, the groups that interact with external hardware or other software systems, and the groups that carry out auxiliary administration and maintenance tasks.
- A generalization/specialization hierarchy should be used to group comparable actors, and it is often helpful to assign one archetype to each similar actor in order to improve comprehension.
- To describe that street from announcement from each actor a the system beneficial cases, population one uses a sag diagram with these players as the Y actors.

The graphic, for instance, depicts the setting of a credit card validation system with a focus on the system's actors. Two different categories of clients can be found (individual client and corporate client). The roles that people take on when interacting with the system are these actors. Other institutions are also represented by actors in this scene, including the Retail Institute (where a customer uses a credit card to make a purchase) and the Sponsoring Financial Institution (which serves as a clearing house for credit card accounts). These two participants are probably software-intensive systems in the real world.



**Figure 11.6 : Modelling the context of a system**

The context of a subsystem is modelled using the same method. A smaller system that is part of a bigger system that is at a higher level of abstraction is frequently referred to as a subsystem. Therefore, when creating systems with interconnected systems, modelling the context of a subsystem is helpful.

**Modeling the requirements for a system :**

A need is a system's design element, characteristic, or behaviour. When you state a system's needs, you are asserting a contract that has been made between external factors and the system itself and that outlines the tasks you expect the system to perform. He mostly doesn't care how the system works; he just wants it to work. A well-behaved system will faithfully, predictably, and dependably fulfil all of its criteria. Although awareness of these criteria will definitely advance as the system is implemented iteratively and step by step, it is crucial to start with an agreement on what the system should do. Similar to this, in order to operate a system effectively, it is crucial to understand how it behaves before purchasing it.

Requirements can be represented in a variety of forms, from free-form text to formal language representation and everything in between. Use case diagrams from UML are vital for handling these needs since they may be used to express the majority, if not all, of a system's functional requirements.

To model the requirements for a system,

- To build the connection between the system and the actors that surround it.
- Take into account each actor's conduct in relation to each wait or request made of the system.
- First name These Usage scenarios Common behaviour.
- Factor To extend additional principal line floating, factor variant behaviour in new use cases; common behaviour inside new use cases is employed for other purposes.

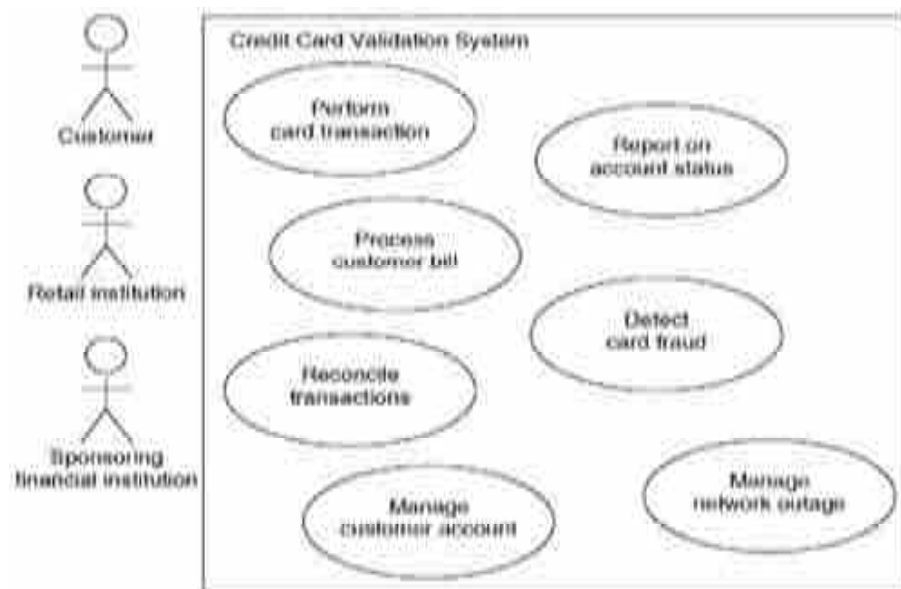


**Object Oriented  
Analysis and Design**

- Template The use case diagram includes these use cases, actors, and their relationships.
- They have use cases that, to a certain extent, do not meet the requirements; you are able to tie some of these to the system as a whole.

The earlier use case diagram is expanded in the figure. While it eliminates the connections between actors and use cases, it also introduces new use cases that are relatively hidden from the typical customer but have important systemic behaviours. This chart is useful because it offers end users, subject matter experts, and developers a standard starting point for visualising, describing, creating, and documenting their choices regarding the functional needs of this system. For instance, detecting card fraud is a crucial action for both the sponsoring bank and the retail institution.

Another behaviour that the system of the various entities in their context requires is the information on the statement of account.



**Figure 11.7 : Modelling the requirements of a system**

The Manage Network Interruption Utility case models a demand that differs differently from the others in that it depicts secondary system behaviour required for the system's sustained, reliable operation.

**❑ Check Your Progress – 2 :**

- \_\_\_\_\_ diagram identify high level services provided by the system  
a. Activity      b. Class      c. Use case      d. Object
- \_\_\_\_\_ shape is used to connect the initial actor to the use case.  
a. circle      b. ellipses      c. line      d. arrow

**11.4 Activity Diagrams :**

**Terms and concepts :**

An activity flow chart demonstrates how one activity leads to the next. A is a state machine's non-nuclear execution path. The final product of the activities is an action, which consists of executable atomic calculations that affect the system's state or return a value. Actions can be pure calculations like evaluating an expression, invoking another operation, sending a signal,

creating or deleting an object. An activity graph visually consists of a number of angles and arcs.

**Common features :**

An activity chart is merely a particular kind of chart that has the same name and graphic information that is a projection onto a model as all other charts. An interaction chart's content is what sets it apart from all other chart kinds.

**Contents :**

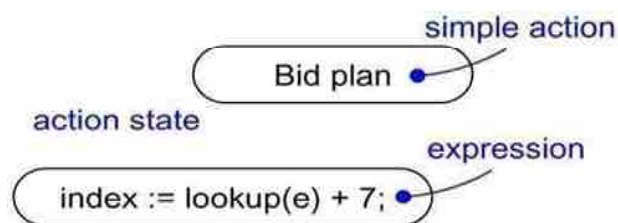
Activity charts commonly contain

- States of activity and states of action
- transitions
- Objects

Like all charts, activity charts can contain notes and restrictions.

**Action States and Activity States :**

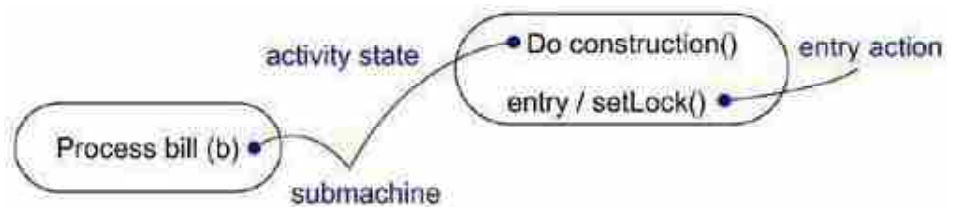
Things take place in the control flow described by an activity diagram. Any expression that either returns a value or indicates the value of an attribute is evaluable. As an alternative, you can build or destroy an object or call an operation on it. You can also give an object a signal. Because they are the states of the system that each represent the execution of an action, these executable atomic calculations are known as action states. You depict a style of activity using a diamond shape, as seen in the image (a symbol with a horizontal top and bottom and convex sides). You can write any expression in that module.



**Figure 11.8 : Action States**

There is no way to separate the modes of activity. Additionally, because the action states are atomic, no interruptions to their functioning can result from external occurrences. Last but not least, it's commonly accepted that work in progress requires very little lead time.

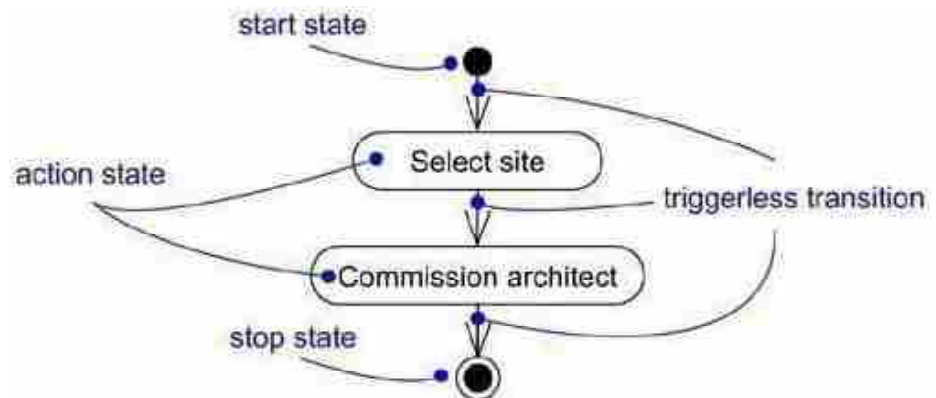
As an alternative, the activity states can be further subdivided, and various activity diagrams can be used to describe their activity. Also, non-atomic, activity states can be interrupted and are typically thought to take some time to complete. A mode of action might be considered a specific case of a mode of activity. A mode of action is an activity that can't be further subdivided. A mode of activity is analogous to a compound whose flow of control is made up of various modes of activity and modes of action. Another activity graph can be found by zooming in on the specifics of an activity mode. As shown in the figure, there is no distinction between a mode of action and a mode of activity, with the exception that a mode of activity may include extra components like secondary machine specifications, input and output actions, and actions that are involved in entering and leaving the mode, respectively.



**Figure 11.9 : Activity States**

**Transitions :**

The flow of control immediately shifts to the following action or mode after the activity in one mode is finished. The path from one action or activity mode to the next is visualised by utilising transitions to specify this flow. As seen in the illustration, a transition in UML is represented as a straightforward straight line.

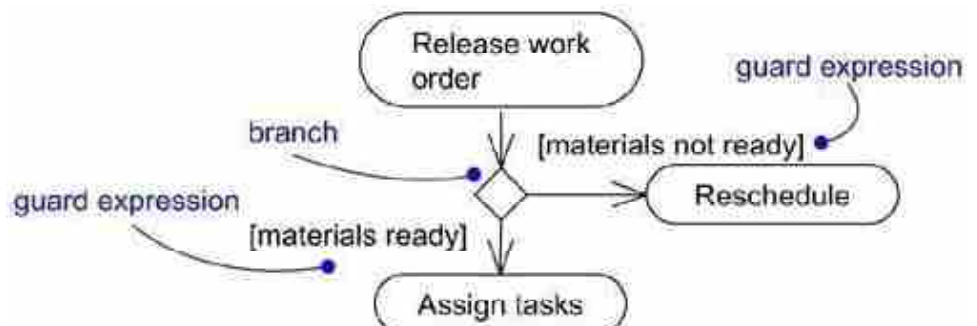


**Figure 11.10 : Trigger less Transition**

Undoubtedly, a control flow needs to have a beginning and an end (unless, of course, it is an infinite flow, in which case it will have a beginning but not an end). Consequently, as the picture illustrates, it is feasible to provide both a stop mode and this start mode (a solid ball) (a solid ball inside a circle).

**Branching :**

Although straightforward sequential transitions are frequent, they are not the only kind of path required to represent the flow of control. You can insert a branch that details alternative routes followed based on a Boolean expression, similar to a flowchart. You depict a branch like a diamond, as the illustration illustrates. One incoming transition and two or more outbound transitions are permitted for a branch. It inserts a Boolean statement on each outbound transition, which is only evaluated once upon branch entry. The guards must not overlap on any of these output transitions because doing so would make the control current unclear, and they must instead cover all potential scenarios (otherwise the control current would be frozen).



**Figure 11.11 : Branching**

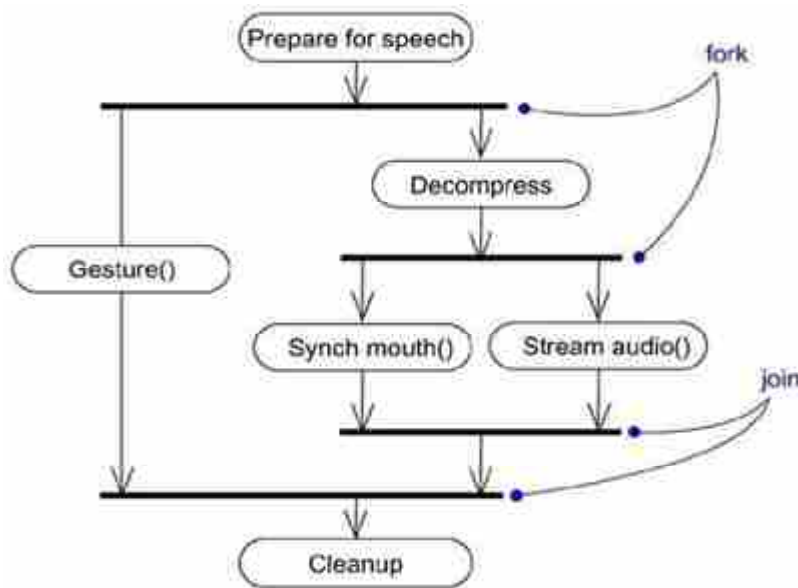
For ease of use, you can mark an outbound transition that denotes the course taken if no other protection terms are discovered to be true by using the else keyword. Using an action mode that displays the value of an iterator, another action mode that increases the iterator, and a branch that determines whether the iteration is finished, you can simulate iteration.

**Forking and Joining :**

The most frequent transitions in activity charts are simple and branching sequential ones. But particularly when modelling commercial processes, you could run into multiple currents at once.

To describe the branching and merging of these simultaneous control streams in UML, utilise a sync bar. The visual representation of a sync bar is a thick horizontal or vertical line.

Think about the several currents that are present when an audio–animatronic entity is controlled to mimic human speech and motions. A fork in the illustration symbolises the splitting of a single control flow into two or more concurrent control flows. A rung may have one input transition and one to several output transitions, each of which denotes a different control flow. The activities connected to each of these pathways continue simultaneously beneath the crossroads. Though these flows in a running system may be sequential but interleaved (in the case of a multi–node system) or actually simultaneous (in the case of a multi–node system), conceptually, the activities of each flow are truly contemporaneous, merely giving the appearance of true contemporaneity.



**Figure 11.12 : Forking and joining**

A join symbolises the synchronisation of two or more concurrent control flows, as the illustration also demonstrates. A join connection may have one outgoing transition and two or more inbound transitions. The actions connected to each of these routes continue side by side through the intersection. Concurrent flows are synchronised upon join connection, which implies that each one waits for all incoming flows to arrive at the join connection before a flow of control continues throughout the connection.

**Swimlanes :**

You will find it advantageous to organise the activity states in an activity diagram according to the business organisation that is in charge of them,

## Object Oriented Analysis and Design

especially when modelling business processes. Because each group in the UML diagram is visually separated from its neighbour by an uninterrupted vertical line, as illustrated in the image, each group is referred to as a trajectory. A path points to a location for an activity.

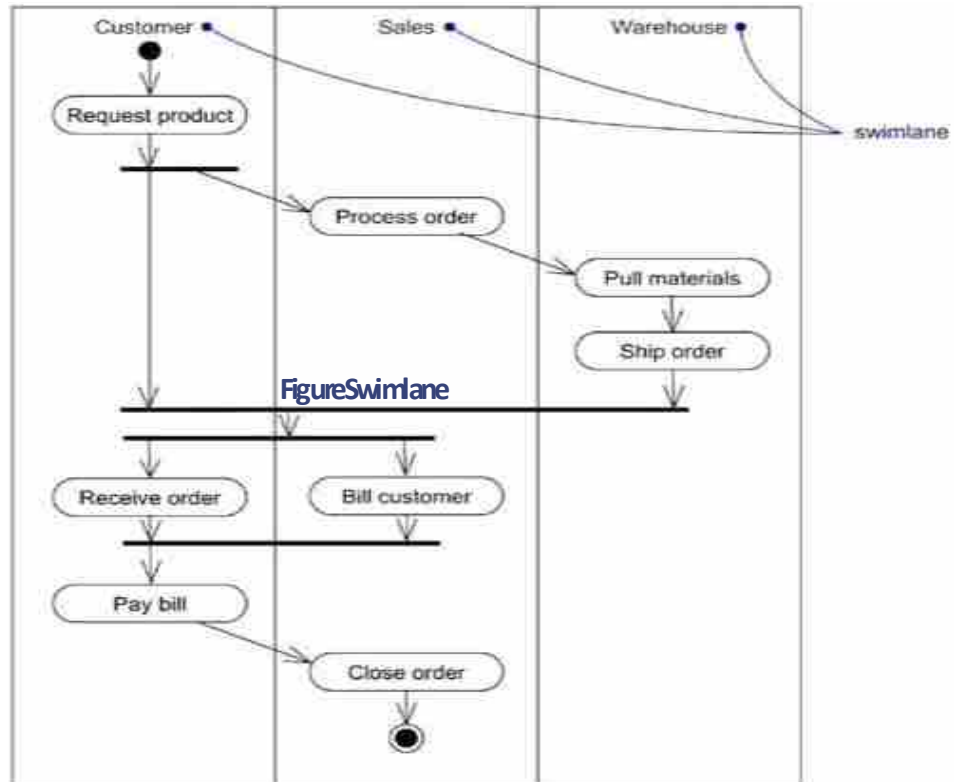


Figure 11.13 : Swimlanes

In its table, each course is given a different name. A street can represent an actual entity, although it doesn't have particularly complex semantics. In an activity graph, each lane indicates a high-level responsibility for a component of the overall activity, which may be carried out by one or more classes. Each activity in a pool activity table is specific to a single pool, but transitions can cross swim lanes.

### Object Flow :

The flow of control connected to an activity diagram can include objects. Your issue space vocabulary will also include classes like Order and Invoice in the order processing workflow shown in the above diagram, for instance. Some actions will result in instances of these two classes (processing an order, for instance, will result in the creation of an order object), while other actions may modify these objects (for example, Submit Order will change the status of the order object to completed).

As seen by the figure, you may define the components of an activity graph by putting the objects there and associating them with a dependency on the activity or transition that creates, deletes, or alters them. Because it shows how an object participates in a control flow, this application of object-to-dependency relationships is known as an object flow.

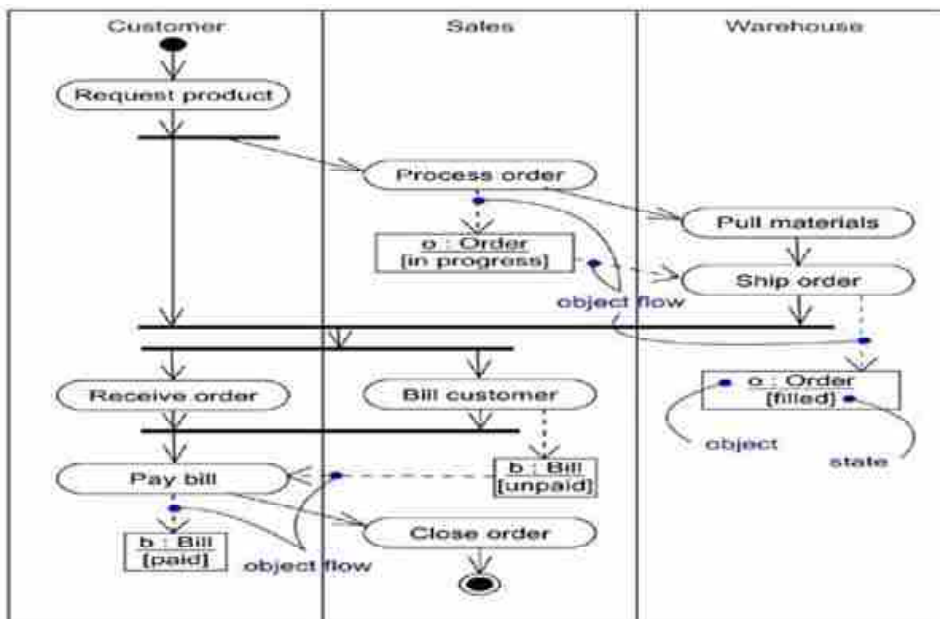


Figure 11.14 : Object Flow

An activity graph can be used to represent an object's flow as well as how its function, status, and attribute values change. As seen in the picture, you name an object's state in parentheses next to the name of the object to represent the object's current state. The value of an object's properties can also be represented by showing them in a space beneath the object name.

**Common uses :**

Activity charts can be used to model a system's dynamic elements. Any abstraction, including active classes, interfaces, components, and nodes, can be active in any view of a system's architecture as part of these dynamic characteristics.

An activity chart can be used in conjunction with any modelling element to represent a dynamic component of a system. Activity graphs are often used to represent a process, class, subsystem, or the system as a whole. In order to depict a situation, you may additionally include activity diagrams for use cases and collaborations (to model the dynamic aspects of a community of objects).

Activity graphs are commonly used in two ways when simulating a system's dynamic elements.

**1. ONE template-one workflow :**

Here, you'll concentrate on the actions that the actors interacting with the system witness. Workflows are used to view, specify, design, and document business processes involving the system that is being developed. They are frequently found on the periphery of software-intensive systems. Especially crucial in this application of activity graphs is object flow modelling.

**2. To model an operation :**

Here, you will model the specifics of a calculation using activity diagrams like flowcharts. The modelling of branch, fork, and join modes is crucial in this use of activity diagrams. The operation parameters and its local objects are included in the context of an activity diagram when it is utilised in this manner.

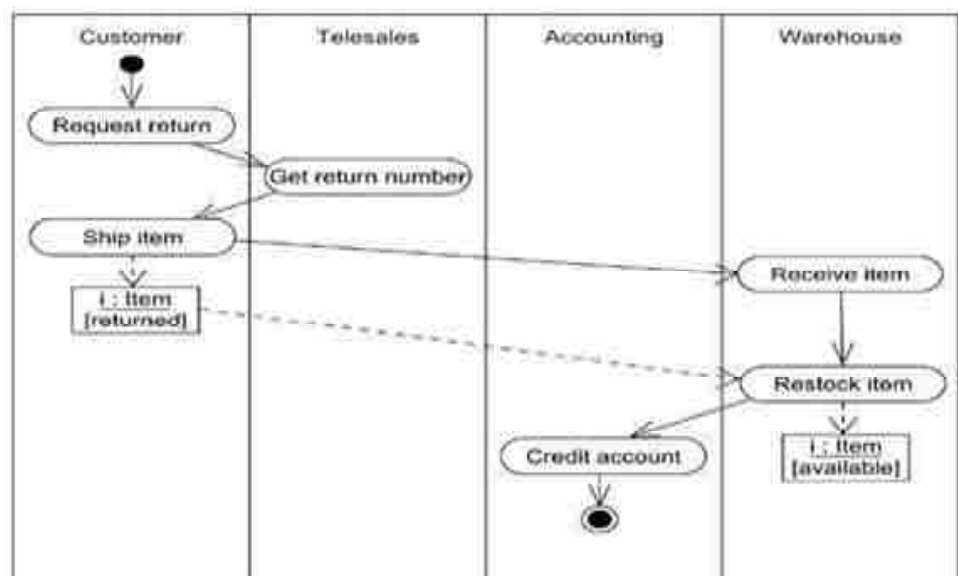
➤ **Common modeling techniques :**

**Workflow modelling :**

To model a workflow,

- Decide on the workflow's focus. It is impossible to depict all interesting workflows in a single diagram for non-trivial systems.
- Pick the business objects that, on a high level, control particular aspects of the overall workflow. These could be more abstract concepts or actual words from the system's vocabulary. In either case, you must make a path for every significant business object.
- Identify the preconditions for the workflow's initial state and the post-conditions for its final state. To assist you in defining the parameters of your workflow, this is crucial.
- Begin with the workflow initiation mode, describe the ongoing activities, and display them as either actions or activities in the activity graph.
- Provide a separate activity chart that grows each for sophisticated tasks or for series of actions that appear more than once. Hiding such actions in activity modes.
- It stands for the changes between these active and passive phases. Your process should begin with sequential flows, then you should think about branching, and only then should you think about branching and joining.
- Recreate any significant objects that are part of the workflow in the activity graph as well. To convey your intentions to the stream of objects, visualise your changing values and be specific as necessary.

The graphic, for instance, depicts an activity chart for a retail business, illustrating the process required when a consumer returns an item by mail order. The work starts with the client's request for a return, moves through Telesales to request a return number, loops back to the customer for the item in shipment, moves to the warehouse to receive and keep the item, and finally concludes with accounting (Credit account). An important object I an instance of an item), as shown in the diagram, also communicates the process and transitions from the returned state to the available state.

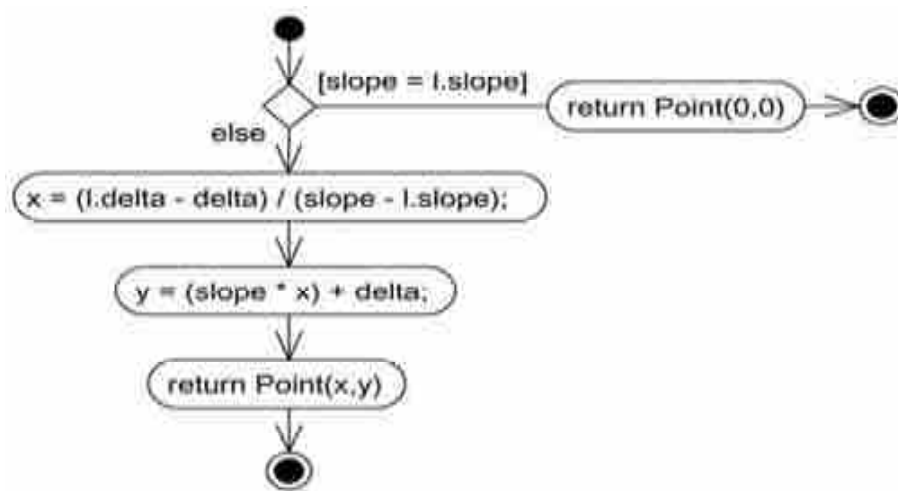


**Figure 11.15 : Modelling a Workflow**

**Modeling of an operation :**

To model an operation,

- Gather the abstractions necessary for this process. The characteristics of the include class, certain nearby classes, the operation arguments, including the return type, if any, are all included.
- Identify the pre-conditions in the operation's beginning state and the post-conditions in its final state. Also note any invariant class versions that the action must satisfy.
- Begin with the operation's initial state, describe the ongoing activities and actions, then add the resulting activity states or action states back to the activity diagram.
- When defining conditional pathways and iterations, use branching.
- Only use branches and merge connections to define parallel control flows if this action is controlled by an active class.



**Figure 11.16 : Modelling an operation**

For instance, the figure depicts an activity diagram for the intersection operation in the context of the Line class, whose signature comprises a parameter (l, an in-parameter for the Line class), and a value of return (from the Point class). The slope and delta attributes of the line class are the two relevant attributes (which contains the displacement of the line from the origin).

As may be seen in the activity diagram that follows, the algorithm is straightforward. The protector first determines whether the slope of the current line and the slope of parameter l are equal. In this case, a point at (0, 0) is returned and the lines do not meet. Otherwise, since x and y are already local objects for the operation, the operation first determines an x value for the intersection point before determining a y value. A point at (x, y) is finally returned.

**□ Check Your Progress – 3 :**

1. System behaviour can be modelled using \_\_\_\_\_ diagram.
  - a. activity
  - b. class
  - c. object
  - d. use case
2. To model business workflow which diagram is best suited ?
  - a. interaction
  - b. activity
  - c. object
  - d. use case



### **11.5 Let Us Sum Up :**

In this unit, we learned what the unified modeling language is and the importance of modeling for application development. We also learn four main patterns with explanation. We also discussed object-oriented modeling with two different approaches to developing any model.

We also discussed the conceptual model of UML and the detailed explanation of its element such as things (object-oriented parts of UML), reporters and diagrams used in UML. Finally, we also discussed the UML architecture.

In this unit you learnt that Behavioural Modelling is an approach used by companies to better understand and predict consumer actions. We have also see that Behavioural modelling uses available data on consumer and business spending to estimate future behaviour in specific circumstances.

We have also learned that the purpose of the use cases is to express user requirements in a technology independent way using a structured narrative form that both the non-IT user and the business analyst can understand. We also learned that use cases have also provided a simple yet powerful structure that provides valuable information about the users of the system and how they are using it.

We also learned the Activity diagram which shows the order of activities in a process, including sequential and parallel activities, and the decisions made. We also discuss that an activity chart is usually created for a use case and can show different possible scenarios.

### **11.6 Answers to Check Your Progress :**

**Check Your Progress 1 :**

1 : a            2 : b

**Check Your Progress 2 :**

1 : c            2 : d

**Check Your Progress 3 :**

1 : a            2 : b

### **11.7 Glossary :**

1. **Activity Diagram :** It shows the order of activities in a process, including sequential and parallel activities, and the decisions made.
2. **Use Case :** A use case is a description of a set of sequences of actions, including variants that a system performs to produce an observable result of value for an actor.
3. **Node :** A node is a physical element that exists while driving and represents a computing resource.

### **11.8 Assignment :**

1. What are the 4 main components of a use case diagram ? Explain.

### **11.9 Activities :**

1. Study the symbols used to draw the use case diagram. Also discuss the purpose of use case diagram.

**11.10 Case Study :**

1. Study the difference of use case diagram and activity diagram.

**11.10 Further Readings :**

1. Maitri Jhaveri, Madhuri B. Arhunshi – Structured and object-oriented analysis and design methodology – person
2. UML for Java programmers by Robert C. Martin

**UNIT STRUCTURE**

- 12.0 Learning Objective
- 12.1 Introduction
- 12.2 Events and Signals
- 12.3 State Machines
- 12.4 Processes and Threads
- 12.5 Time and Space
- 12.6 State Chart Diagrams
- 12.7 Let Us Sum Up
- 12.8 Answers for Check Your Progress
- 12.9 Glossary
- 12.10 Assignment
- 12.11 Activities
- 12.12 Case Study
- 12.13 Further Readings

**12.0 Learning Objectives :**

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

- About advanced behavioural modelling
- About Events and Signals
- About State Machines
- About Processes and Threads
- About Time and Space
- About State Chart Diagrams

**12.1 Introduction :**

Businesses utilise behavioural modelling as a strategy to comprehend and forecast consumer behaviour better. Utilizing data on consumer and commercial spending, behavioural modelling predicts future behaviour under particular conditions. Financial institutions use behavioural modelling to calculate the risks of lending money to a person or company, while marketing companies use it to target advertisements.

A specific instance of a crucial truth with a time and place is called an event. Asynchronous occurrences include a signal, the passage of time, and a change in state. The state machine diagram, also known as a state diagram or a state transition diagram, depicts the sequential states that an object experiences while moving through the system.

**12.2 Events and Signals :**

**Terms and Concepts :**

An event is the description of a noteworthy occurrence that takes place in both time and space. Inside An event is the occurrence of a stimulus that can cause a state transition in the context of state machines. A signal is a special kind of event that shows how an asynchronous stimulus specification is sent across instances.

**Types of Events :**

Events might take place internally or externally. Events that happen outside of the system's actors are referred to as external events. External events include things like a button press and the stoppage of a crash sensor. Internal events are those that take place between system-dwelling items. An illustration of an internal event is an overflow exception.

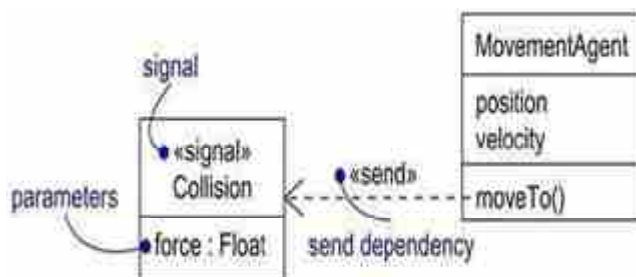
Four different sorts of events can be modelled in UML: signals, calls, time passing, and state changes.

**Signals :**

A signal is a named object that is sent and received asynchronously by two different objects. Exceptions are the most typical internal signal you need to model and are supported by the majority of modern programming languages.

Simple classes and signals share many similarities. Signals, for instance, can contain instances, though it is typically not essential to explicitly model them. In generalisation relationships, signals can also play a role. This enables you to represent hierarchies of events, some of which are general and some of which are special. Signals can also have properties and operations in terms of classes.

A state transition in a state machine or the transmission of a message during an interaction can both send a signal. Signals can also be sent by performing an operation. In fact, defining the signals that an element's actions can transmit when modelling a class or interface is crucial to defining the behaviour of the element. A dependence relationship, referred to as sending, is used in UML to model the connection between an action and the events it can send.

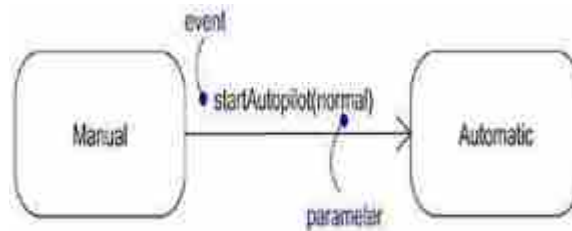


**Figure 12.0 : Signals**

**Call Events :**

A call event indicates the transmission of an operation, much as a signal event describes the occurrence of a signal. The event may cause a state transition in a state machine in either scenario. A call event is often synchronous, but a signal is an asynchronous event. As a result, if one object that has a state machine conducts an operation on another object, control is passed from the

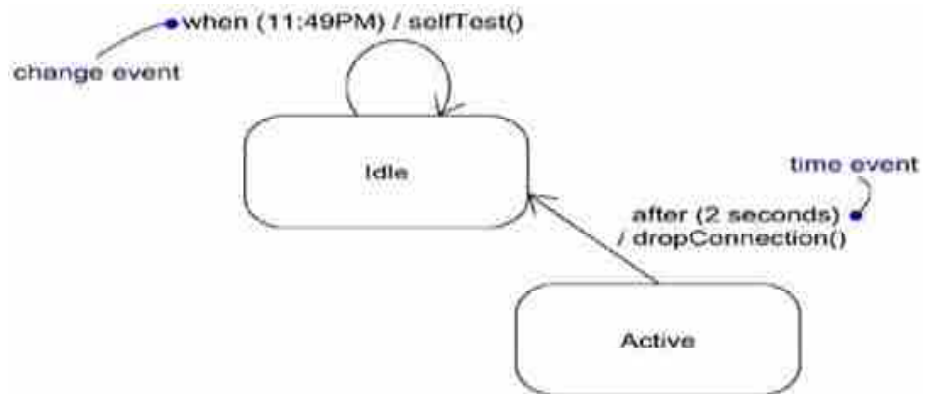
sender to the recipient, the event initiates the transition, the action is finished, the recipient enters a new state, and control is then returned to the sender.



**Figure 12.2 : Call events**

**Time and Change Events :**

An occasion that symbolises the passage of time is a time event. According to the figure, one can temporarily use the after keyword in UML by doing so after a statement that is analysed over time. These expressions may be straightforward, such as after two seconds, or sophisticated, such as after one millisecond from the end of inactivity. The start time of that expression is the moment it assumed its current state, unless you specify otherwise.



**Figure 12.3 : Weather and Change events**

An event that signifies a change in status or the fulfilment of a condition is referred to as a "change event." A change event in UML is depicted using the keyword followed by a Boolean expression, as seen in the picture. Such expressions can be used to indicate an exact time, such as when the time is 11:59, or to test an expression repeatedly, as when the height is below 1000 feet.

**Sending and Receiving Events :**

Both signal events and call events include a minimum of two objects: the object sending the signal or calling for the action and the item being targeted by the event. The semantics of events intersect with the semantics of active objects and passive objects due to the asynchronous nature of signals and the fact that asynchronous calls are signals in and of themselves.

Any instance of any class has the ability to signal an object or call an action on a target object. When an object transmits a signal, it does so without waiting for a response from the receiver and immediately resumes its control flow.

**❑ Check Your Progress – 1 :**

1. Which of the following are used to model records of activities that describe what happened in the past/what need to be done later.
  - a. Signal
  - b. Events
  - c. Node
  - d. Object

**12.3 State Machines :**

**Terms and Concepts :**

A state machine is a behaviour that describes the successions of states that an object experiences throughout the course of its lifespan in response to events, as well as how that object reacts to those events. A condition is satisfied, an action is carried out, or an event is anticipated by an item in a state. A specific instance of a crucial truth with a time and place is called an event. An event is the occurrence of a stimulus that can cause a state transition in the context of state machines.

A transition is a relationship between two states that signals that when a specific event occurs and certain conditions are satisfied, an object in the first state will carry out specific actions and enter the second state. A state machine activity is a non-atomic execution that is now taking place. A model's state might change as a result of an action, which is an executable atomic calculation that returns a value. A state is graphically shown as a rectangle with rounded corners. A solid straight line is used to symbolise a transition.

**Context :**

Everything has a lifespan. An object is formed during creation, and it no longer exists after being destroyed. On rare occasions, in addition to being the target of an action, an object can also act on other objects by sending them messages (being the target of a message). These messages will frequently be straightforward synchronous operation calls. For instance, a customer class instance might use the `getAccountBalance` operation on a `BankAccount` class instance. Because their present behaviour is independent of their past, objects like these don't require a state machine to define their behaviour.

**States :**

A state is a circumstance or position in an object's life where it satisfies a requirement, carries out an action, or waits for a happening. A limited amount of time is spent in a state by an object. For instance, a heater in a house can be in any one of four states: inactive (waiting for an order to begin heating the house), activating (gas is on, but temperature is not yet high enough), active (gas and fan are on), and shutting down (the gas is off but the fan is on, pushing the waste heat out of the system).

An item is said to be in that state when its state machine is in that state. For instance, a heater could be off or shut down.

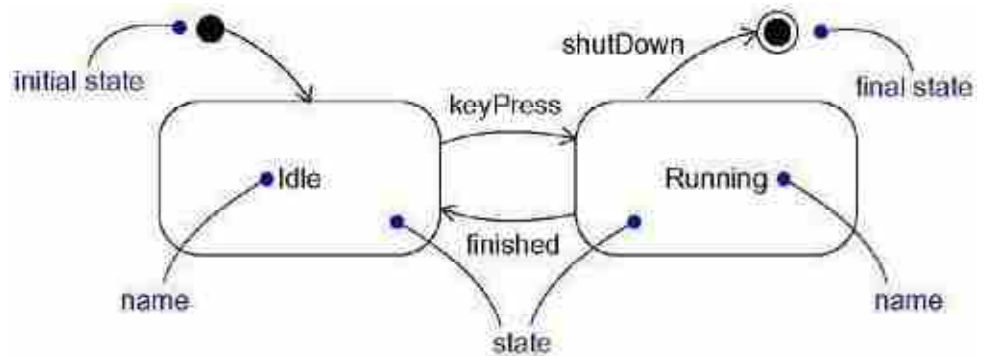
Multiple components make into a state.

**Table 12.1 : State Parts**

Name	A textual string to distinguishes the state from other state; – a state may be anonymous , meaning that it has no name
Entry/Exit Action	Actions executed on entering and exiting the state, respectively
Internal transitions	Transitions that are handled without causing a change in state

Substates	The nested structure of a state, involving disjoint (sequentially active) or concurrent (concurrently active) substates
Deferred	A list of events that are not handled in that state but, rather, are postponed and queued for handling by the object in another state events

As Figure shows you to represent a state as a rectangle with rounded corners.



**Figure 12.4 : V**

**Initial and Final States :**

Two special states can be specified for an object's state machine, as shown in the picture. The state machine's or substate's default beginning point is specified by the initial mode, which comes first. A solid black circle signifies an initial condition. The second is the final state, which denotes that the associated state or state machine's execution is finished. A solid black circle enclosed by an empty circle signifies a finite state.

**Note :**

Pseudo-states are what the initial and final states are in reality. Except for a name, none of them can possess the typical components of a regular condition. The whole complement of functionality, including a guard state and an action, can be present throughout a transition from an initial to a final state (but not a trigger event).

**Transitions :**

A transition is a relationship between two states that signals that when a specific event occurs and certain conditions are satisfied, an object in the first state will carry out specific actions and enter the second state. The transition in such a change of state is called to burn. The object is said to be in the source state up until the transition is initiated, and the target state up until firing. For instance, when a circumstance like too much cold occurs, a heater may turn from off to on (with the desired temperature parameter).

A transition is depicted in the illustration as a straight, solid line going from the source state to the destination state. A transition is referred to as an auto transition if its source and destination states are identical.

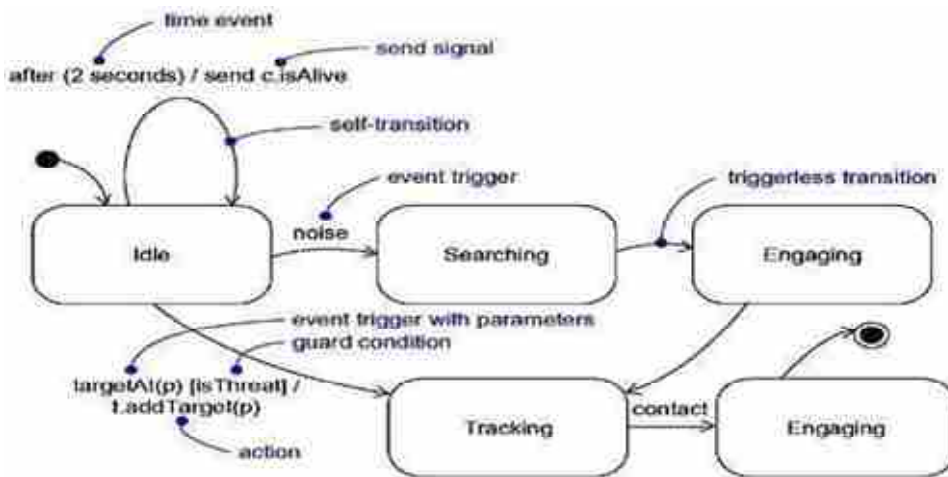


Figure 12.5 : Transitions

**Event Trigger :**

A specific instance of a crucial truth with a time and place is called an event. An event is the occurrence of a stimulus that can cause a state transition in the context of state machines. Events can be signals, calls, the passing of time, a change in state, or any of the other things depicted in the above diagram. State expressions and guard actions are examples of parameters that can have transitionable values in a signal or call. Another possibility is an inactivated transition, which is symbolised by an inactivated transition following an event. When its source state is present and its action has been completed, a triggered less transition, also known as a completion transition, is implicitly triggered.

**Guard :**

A guard condition is shown as a Boolean statement enclosed in square brackets and inserted after the trigger event, as shown in the above picture. A protection state is only assessed after the incident that causes its shift has taken place. As long as these conditions do not overlap, it is therefore conceivable to have numerous transitions with the same event trigger and from the same source state.

When an event occurs, a guard condition is only ever examined once for each transition; however, if the transition is triggered again, the guard condition may be evaluated again. Conditions for an object's state can be included in the Boolean expression (for example, the expression `aHeater at idle`, which evaluates to True if the heating object is currently idle).

**Action :**

An action is an atomic computation that can be executed. Actions involve sending signals to objects, constructing or destroying other objects, including the object that owns the state machine, as well as calling operations on other visible objects. The signal name is preceded by the term `send` as a visual signal, as seen in the above figure, which is a particular notation for transmitting signals.

**Advanced States and Transitions :**

Only the fundamental capabilities of UML modes and transitions are necessary to model a wide range of behaviours.



By utilising these features, you will create flat state machines, whose behavioural patterns will be limited to arcs (transitions) and vertices (states).

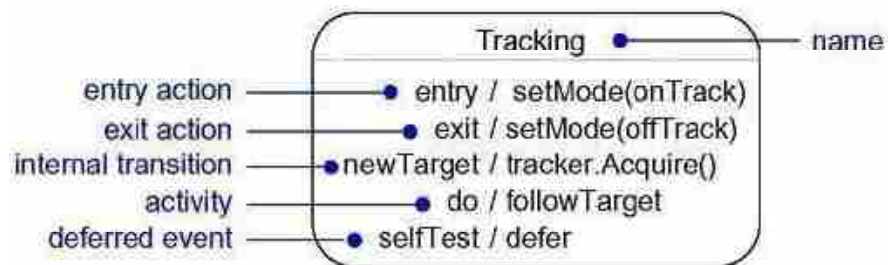
But UML state machines contain a number of sophisticated capabilities that let you control intricate behavioural patterns. When compared to flat state machines, these properties frequently reduce the number of modes and transitions needed and encode a variety of frequent and relatively difficult idioms. Input and output actions, internal transitions, activities, and deferred events are a few of these advanced characteristics.

**Entry and Exit Actions :**

You should transmit the identical action each time you enter a state in a variety of modelling scenarios, independent of the transition that brought you there. Similarly, regardless of the transition that drew you out of a state, you want to carry out the same activity when you leave it.

For instance, you might wish to expressly indicate that a missile control system is tracking when it is in tracked mode and off track when it is in out-of-track mode. This effect can be produced using flat-state machines by appropriately configuring these actions on each input and output transition. It is a little misleading, though, because you have to remember to include these actions each time you add a new transition. You must tap on each neighbouring transition in order to edit this action.

The image illustrates how UML offers an acronym for this phrase. Along with the relevant action, the status token may also contain an input action (designated with the keyword event input) and an output action (designated with the keyword event output). Your entry action is sent each time you enter the state, and your exit action is sent each time you leave the state.



**Figure 12.6 : Advanced State and transitions**

**Activities :**

An object is normally idle while it waits for an event to happen when it is in a state. However, there are situations when you would want to imitate an ongoing company. When an object is in a state, it works and will keep working until something happens to stop it. For instance, as long as an item is in the Follow mode, it can follow the Target. The specific transition does in UML is used to define work to be done in a state after submission of the input document, as the graphic illustrates. Another state machine might be mentioned in a do transition activity (such as followTarget). Additionally, you may provide a chain of events. For instance, do/op1(a), op2(b), and op3 (c). Sequences of actions can be stopped, but individual actions never are. Events can be handled by the related state between each action (separated by semicolons), resulting in a transition out of state.

**Deferred Events :**

Think about a state like tracking. Assume there is just one transition out of this state, as indicated in the figure, and it is caused by the event switch. All contactless events that are not handled by their sub-modes during discovery mode will be lost. This suggests that even though the event might happen, it will be delayed, and nothing will be done as a result of its existence. In any scenario in the model, you want to pay attention to some events while ignoring others. Include the ones you wish to recognise as transition event triggers; just omit the ones you want to disregard. In some modelling circumstances, though, you might want to acknowledge some events while saving your reaction for later. You might want to postpone your response to signals like self-tests, which may have been delivered by a system health agent, during exploration mode, for instance.

Using delayed events, you may define this behaviour in UML. A list of events is called a deferred event when their occurrence in the state is postponed until a condition is met where they are not, at which point they take place and can cause transitions as if they had just happened. You can indicate a deferred event by specifying the event with the deferred special action, as seen in the above figure. In this illustration, self-test events are possible in discovery mode, but they don't stop until the object enters activation mode, at which point they seem to have recently happened.

**□ Check Your Progress – 2 :**

1. Which of from the following can model the behaviour of an individual object ?  
a. Activity      b. Class      c. State Machine      d. Use case

**12.4 Processes and Threads :**

**Terms and Concepts :**

An object is considered to be active if it controls a process or thread and can initiate monitoring activities. A class whose instances are active objects is said to be an active class. A process is a large flow that can operate alongside other processes in parallel. A thread is a small flow that can operate alongside other threads in the same operation. A rectangle with thick lines is used to graphically depict an active class. Processes and threads are portrayed as active classes with preconceived notions (and also appear as sequences in interaction diagrams).

**Flow of Control :**

There is a control flow in a system that runs only sequentially. This indicates that just one thing can occur at a time. A sequential programme starts with the control rooted at the beginning and sends each operation one at a time. A sequential programme will only process one event at a time, queuing or removing any concurrent external events, even if they happen between actors outside the system. It is called control flow for this reason. A sequential program's execution site moves from one statement to the next in sequential order, as can be seen if you monitor it. If there is any recursion or iteration, you will watch the flow loop on itself. The actions may branch, scroll, and jump. However, there would only be one flow of execution in a sequential system.

In a simultaneous system, there are multiple control flows, allowing for multiple events to take place concurrently. There are multiple concurrent control flows in a concurrent system, each of which is anchored in the head by a different process or thread. While a concurrent system is active, you can logically observe many execution locations if you take a picture of it.

In the UML, a process or thread that is the source of a distinct control flow and that is on the same level as all other control flows is represented by an active class.

#### **Classes and Events :**

Although they have a very unique property, active classes are simply classes. A normal class does not reflect an independent flow of control, whereas an active class does. Simple classes, as contrast to active classes, are implicitly referred to as passive because they are unable to initiate the control activity on their own.

Each independent control flow is given a name as it models concurrent systems with active objects. A flow of control that corresponds to the creation of an active object begins, and it stops when the item is destroyed.

The same properties apply to all classes, including active classes. There may be instances for active classes. Attributes and operations are possible for active classes. Active classes can engage in association, generalisation, and dependence interactions (including aggregation). Any UML extension technique, such as stereotypes, tagged values, and constraints, can be used by active classes. Interfaces can be implemented by active classes. Collaborations provide for the realisation of active classes, and state machines allow for the specification of an active class behaviour.

Anywhere that passive items are shown in a diagram, active things may also be shown. Interaction diagrams can be used to model cooperation between active and passive objects (including sequence and collaboration diagrams). On a state machine, an active item can be thought of as the event's intended target.

Both passive and active objects can transmit and receive signal events and call events in state machines.

#### **Standard Elements :**

Active classes are compatible with all UML extension techniques. The majority of the time, you will use tagged values to expand the active class's characteristics, such as defining its scheduling policy.

When a process is bold, it signifies that it is something that the operating system is aware of and that it is running in a different address space. Each programme runs as a process in its own address space on the majority of operating systems, including Windows and Unix. Generally speaking, each process on a node is equal to the others and engages in competition for the same resources that are present on the node. Processes never integrate with one another. True concurrency is feasible on a node with many processors. If a node only has one processor, the underlying operating system only provides the appearance of genuine concurrency.

A thread weighs little. Even the operating system itself might be aware of it. It frequently runs within the linked process's address space and is concealed in a larger, heavier process. A thread is a member of the Thread class in Java, for instance. As competitors for the same resources accessible

in the process, all threads residing inside the context of a process are on an equal footing with one another. Threads are never woven into one another. Because the operating system of a node schedules processes rather than threads, there is typically just the illusion of actual concurrency between threads.

**Communication :**

Sending signals from one object to another is how objects communicate with one another. There are four possible interaction combinations in a system with active and passive items that you need to take into account.

First, passive objects can communicate with one another by sending messages. That interaction is nothing more than a straightforward invocation of an operation, if that one control current is flowing across these objects at any given time.

Second, messages can be transmitted between active objects. When this occurs, there is communication between processes and two different communication methods are available. First, an action from another object can be called concurrently by an active object. The caller calls the operation, the caller waits for the recipient to accept the call, the action is invoked, an object (if any) is returned to the caller, and then the two proceed on their individual paths. This type of communication is known as encounter semantics. The two control flows are synchronised during the call. Second, an active object has the ability to call an action on another object or transmit a signal asynchronously. The caller transmits the signal or calls the operation in this sort of communication, and the operation then continues on its own. The recipient continues on its journey after the signal or call is finished as long as it is ready (with intermediate events or queued calls). Because the two objects are not synchronised and instead one leaves a message for the other, this is known as a mailbox.

As seen in the picture, the UML represents synchronous messages as full arrows and asynchronous messages as half arrows.

Third, an active object can transmit a message to a passive object. If more than one active object sends its flow of control through a passive object at once, a problem develops. The timing of these two flows in such a scenario needs to be very precisely modelled, as will be covered in the following section.

Fourth, an active item can receive a message from a passive one. This may appear to be against the law at first, but if you keep in mind that any flow of control is linked to an active object, you will realise that a passive object sending a message to an active object has the same semantics as an active object forwarding a message. To an active object: Message.

**Process Views :**

A system's process view is displayed, specified, constructed, and documented in large part by active objects. The threads and processes that make up the system's concurrency and synchronisation mechanisms are included in the process view of a system. Performance, scalability, and performance of the system are the main concerns from this perspective. The same chart types used in the project view, such as class diagrams, interaction diagrams, activity diagrams, and state diagrams, are used in UML to depict the static and dynamic features of this view, with an emphasis on active classes to represent these threads and processes.

❑ **Check Your Progress – 3 :**

1. Program in execution is called \_\_\_\_\_  
a. Activity      b. Process      c. Data load      d. Program

**12.5 Time and Space :**

**Terms and Concepts :**

A timestamp serves as a marker for the time of an occurrence. A timestamp is created graphically as an expression of the name given to the message (which is usually different from the action name sent by the message). An expression is considered to be temporal if it is judged against an absolute or relative time value. A semantic assertion regarding the relative or absolute worth of time is known as an once constraint. The graph of a time limit is a string wrapped in square brackets that, in the case of constraints, is typically connected to an element through a dependence relationship. Position refers to a component's location on a node. Graphically, the position is shown as a tagged value, which is a string that is placed under the name of an element and contained in square brackets, or as the presence of components in nodes.

**Time :**

By definition, real-time systems are time-sensitive systems. Events can happen at predictable absolute times or predictable times relative to the event itself; reactions might happen at predictable absolute times or unpredictable times relative to the event itself.

When modelling the time-critical nature of a system using UML, message transmission represents the dynamic part of any system. As a result, you may give each message in an interaction a name to use as a timestamp. Typically, messages in a conversation don't get assigned names. The name of an event, like a signal or a call, serves as their primary representation. Because the same event can cause many messages to be sent, the event name cannot be utilised to create an expression. Use the explicit message name in a timestamp to identify the message you want to mention if the chosen message is uncertain.

The only thing a timestamp is is a phrase. derived from a message's name in a conversation. You can use a message name to refer to any of the message's three components, such as the start time, end time, and driving time. Then, you can use these methods to provide arbitrary complicated temporal expressions, potentially even by employing constants or variables for weights or offsets (provided these variables can be bound while driving). In order to characterise the temporal behaviour of the system, it is also feasible, as shown in the picture, to place these times expressions within a time restriction. You can directly attach limitations using dependencies or indicate them as restrictions by putting them next to the appropriate message.

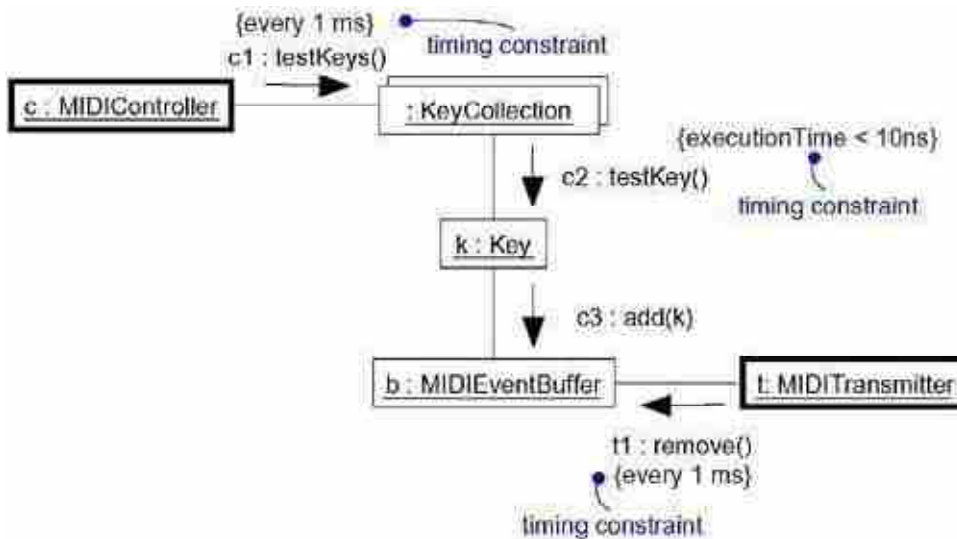


Figure 12.7 : Time

**Location :**

Distributed systems by their very nature have parts that are physically dispersed throughout the system's nodes. Components can migrate from one node to another in some systems, but in many systems, they are repaired as they are loaded.

In Implementation diagrams that depict the topology of the processors and other hardware that the system is running on are used in the UML to describe the visualisation of a system's implementation. These nodes house components like executables, libraries, and tables. Certain components will be present in each instance of a node, and each instance of a component will be owned by only one instance of a node (although instances of the same component type can be spread across different nodes). For instance, as shown in the diagram, the KioskServer node may host the vision.exe executable component.

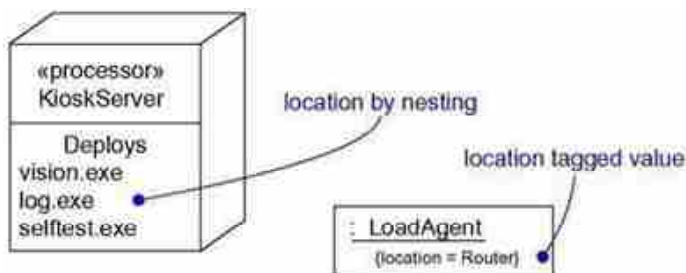


Figure 12.8 : Location

Simple class instances may also exist on a node. For instance, the Router node in the diagram hosts an instance of the LoadAgent class.

There are two approaches to model an element's location in UML, as shown in the image. First, you can physically embed the element (textually or graphically) in an additional space in its enclosing node, as demonstrated with KioskServer. Second, you may identify the node on which the class instance sits by using the location of the defined tagged value, as demonstrated for LoadAgent.

When it's crucial to give a visual cue in your component grouping and spatial separation diagrams, you'll often employ the first shape. The second form is used for modelling a network where an element's position is significant

but not primary, such as when you wish to show messages being passed between instances.

❑ **Check Your Progress – 4 :**

1. In sequence diagram, vertical dimension shows \_\_\_\_\_  
a. line                      b. time                      c. abstract                      d. message

**12.6 State Chart Diagrams :**

**Terms and Concepts :**

A state diagram depicts a state machine and places emphasis on the transfer of control between states. A state machine is a behaviour that describes the series of states that an item experiences during the course of its life in response to events, as well as how the object reacts to those events. A condition is satisfied, an action is carried out, or an event is anticipated by an item in a state. A specific instance of a crucial truth with a time and place is called an event. An event is the occurrence of a stimulus that can cause a state transition in the context of state machines. A transition is a relationship between two states that signals that when a specific event occurs and certain conditions are satisfied, an object in the first state will carry out specific actions and enter the second state. A state machine activity is a non-atomic execution that is now taking place. A model's state might change as a result of an action, which is an executable atomic calculation that returns a value. A state diagram visually consists of a group of vertices and arcs.

**Common features :**

A state diagram is merely a particular kind of diagram that has the same name and graphic content that is a projection onto a model like all other diagrams. A state graph's content is what sets it apart from all other graph kinds.

**Contents :**

State chart diagrams usually cover

- Composite and simple states
- An activity graph is separated from the status graph it includes by transitions, such as events and activities. A state machine in which all or most of the states have been active and where all or most of the transitions are triggered from completion from activities in the State of origin is an activity diagram, which is essentially a projection of the elements present in an activity diagram.

**Common Uses :**

State diagrams are used to model a system's dynamic elements. These dynamic features can include the event-driven behaviour of any class (including active classes), interface, component, or node in any view of the system architecture.

You may model some dynamic characteristics of a system using a state diagram in conjunction with just about any other modelling tool. State diagrams are often used to represent a class, a subsystem, or the entire system. State diagrams may also be affixed to use cases (to model a scenario).

You will often utilise state diagrams in one method when modelling the dynamic components of a system, class, or use case.

**To model reactive objects :**

An entity whose behaviour is best described by how it reacts to events that are sent outside of its environment is said to be reactive or event-driven. Until it gets an event, a reactive object is often inert. Your reaction to an experience is typically influenced by earlier ones. The object returns to sleep after responding to an event and waits for the next one. You will concentrate on the stable states of these kinds of objects, the things that cause them to shift from one state to another, and the things that happen when they do.

**Check Your Progress – 5 :**

1. State chart diagrams are needed when \_\_\_\_\_
  - a. The execution of scenario is to be traced.
  - b. The class has complex life cycle
  - c. Need to allocate classes and objects to modules
  - d. Need to allocate processes to processors

**12.7 Let Us Sum Up :**

In this unit you learned that the behavioural modelling is an approach used by companies to better understand and predict consumer actions. You also learned that behavioural modelling uses available data on consumer and business spending to estimate future behavior in specific circumstances. It is used by financial institutions to estimate the risks associated with providing funds to an individual or business and by marketing firms to target advertising.

You also learned that an event is the specification of an essential fact that has a place in time and space. you have also learned a signal. You also learned that the state machine diagram is also called a state diagram or state transition diagram and it shows the order of states through which an object passes through the system.

**12.8 Answers for Check Your Progress :**

- Check Your Progress 1 :**  
1 : b
- Check Your Progress 2 :**  
1 : c
- Check Your Progress 3 :**  
1 : b
- Check Your Progress 4 :**  
1 : b
- Check Your Progress 5 :**  
1 : b

**12.9 Glossary :**

1. **Event :** An event is the description of a noteworthy occurrence that takes place in both time and space.
2. **Signal :** A signal is a named object that is sent and received asynchronously by two different objects.



3. **State** : A state is a circumstance or position in an object's life where it satisfies a requirement, carries out an action, or waits for a happening.

**12.10 Assignment :**

1. What is behaviour modelling ? Explain Events and Signals.

**12.11 Activities :**

1. Discuss the state machine.

**12.12 Case Study :**

1. What is state chart diagram ? Draw a state chart diagram for online shopping.

**12.13 Further Readings :**

1. Behavior Modeling – Foundations and Applications, publisher: Publisher: Springer International Publishing AG
2. Sanders Emory – Modeling and Simulation of Human Behavior, Publisher: iUniverse

**UNIT STRUCTURE**

- 13.0 Learning Objective
- 13.1 Introduction
- 13.2 Architectural Modelling: Component
- 13.3 Deployment
- 13.4 Component Diagrams
- 13.5 Deployment Diagrams
- 13.6 Let Us Sum Up
- 13.7 Answers for Check Your Progress
- 13.8 Glossary
- 13.9 Assignment
- 13.10 Activities
- 13.11 Case Study
- 13.12 Further Readings

**13.0 Learning Objectives :**

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

- Study about architectural modelling
- Study about component
- Study about deployment
- Study about component diagram
- Study about deployment diagram

**13.1 Introduction :**

An architectural model is a partial abstraction of a system. It is an approximation and captures the different properties of the system. It is a reduced version and is built with all the essential details of the system. Architectural modelling involves identifying system features and expressing them as models so that the system can be understood.

Architectural models allow the display of information about the system represented by the model. The modelling process can be bottom – up / inside–out, where the details of the system are built based on knowledge of the components and interconnections and how they are composed to realize the system's properties. Alternatively, it can be top–down / outside–in, where details about components and interconnections are taken from knowledge of the whole.

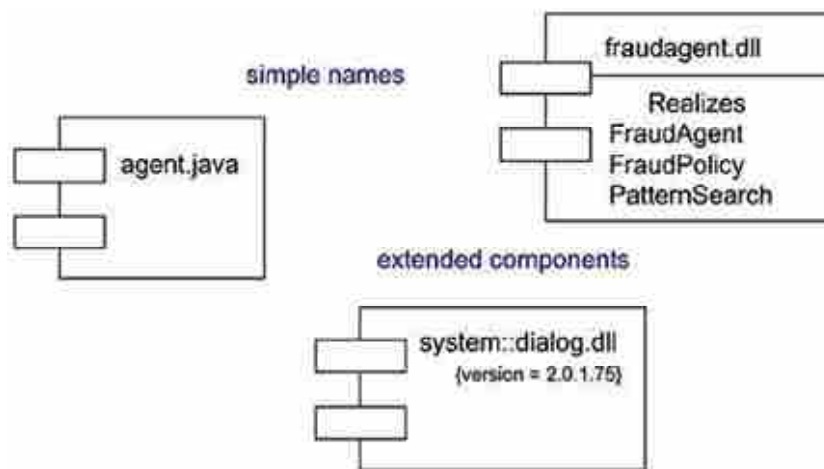
## 13.2 Architectural Modelling : Component :

### Terms and Concepts :

A component is a genuine, replaceable piece of a system that fits and makes a set of interfaces possible. A component is graphically shown as a rectangle with tabs on it.

### Names :

Each component needs a name that sets it apart from the others. A name is a string of text. A path name is the component name followed by the name of the package in which the component is found; a simple name is all that is known about that name. Typically, a component's name and nothing else are displayed, like in the figure. Components can be drawn with marked values or extra spaces to reveal their details, similar to how classes can be drawn.



**Figure 13.1 : Simple and extended component**

### Components and Classes :

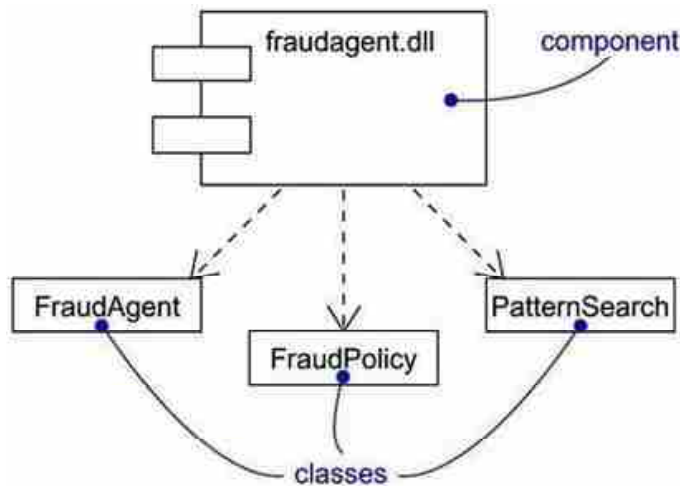
Components and classes share many characteristics, including names, the ability to implement a set of interfaces, the ability to participate in issues of dependence, generalisation, and association, the ability to be embedded, the ability to have occurrences, and the ability to take part in interactions. The distinctions between components and classes do, however, differ significantly in some cases.

- Components refer to tangible objects that exist in the realm of bits, while classes refer to logical abstractions. In a nutshell, whereas components may reside on nodes, classes may not.
- Components are at various levels of abstraction and reflect the physical packaging of, on the other hand, logical components.
- Classes capable of direct operations and properties. components that, in general, have operations that are exclusively accessible through their interfaces

The most significant distinction is the first one. The choice of whether to use a class or a component when modelling a system is straightforward: if the object you're modelling exists directly on a node, use a component; otherwise, use a class.

The second variation points to a connection between classes and components. A component, in particular, is the concrete realisation of a number of other logical entities, such as classes and partnerships. A dependency

relationship can be used to explicitly depict the connection between a component and the classes it implements, as illustrated in the picture. Most of the time, you never need to graphically represent these interactions. Instead, it will continue to include them in the component specification.



**Figure 13.2 : V**

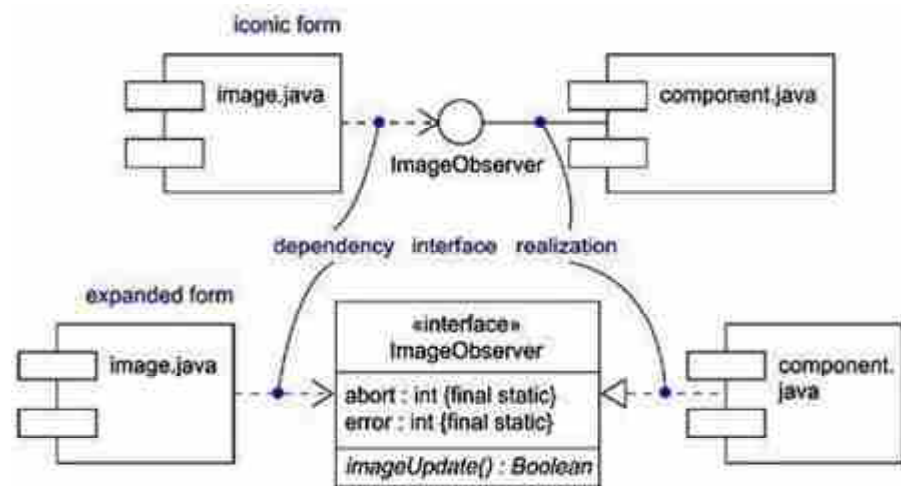
The third distinction focuses on the way interfaces link classes and components. Components and classes can implement interfaces, as is further explained in the following section, but often a component's services are only accessible through its interfaces.

**Components and interfaces :**

An interface is a group of activities used to define a class or component's service. It's crucial to consider how the component and interface are connected. Interfaces serve as the binding agent for all of the most popular component-based operating system installations (including COM +, CORBA, and Enterprise Java Beans).

By describing interfaces that stand in for the major system seams when utilising one of these capabilities, you can deconstruct the physical implementation of the system. Then, it offers both components that carry out the interfaces and other components that make use of the services' interfaces to access other components. You can implement a system with services that are fairly site-independent and interchangeable using this method, as detailed in the next section.

As the graphic illustrates, there are two ways to depict the connection between a component and its interfaces. The interface is portrayed in its classic form in the first (and most popular) style. The interface is connected to the component that realises it through an omitted realisation connection. The second presentation style shows the interface in its enlarged version, possibly revealing how it works. The interface is related to the component that realises it via a full realisation connection. In both situations, the component that uses the interface to access the services of the other component is linked to the interface through a dependence connection.



**Figure 13.3 : Component and Interfaces**

A component's realised interface, or one that it offers as a service to other components, is referred to as an export interface. Many export interfaces can be provided by a single component. An import interface is the interface that a component makes use of; it is an interface that the component adheres to and is based on. An element can adjust to a variety of import interfaces. Interfaces can also be imported and exported by a component.

One component may export a certain interface while importing it into another. The direct dependency between the components is broken by the presence of this interface between the two components. No matter which component implements a particular interface, a component that utilises that interface will function properly. Of course, a component can only be utilised in a context if all of its import interfaces are supported by the export interfaces of other components.

**Binary Replaceability :**

Any component-based operating system installation's primary goal is to make it possible to assemble computers from interchangeable binary components. This implies that a system can be built from its component parts, then developed by adding new parts and removing old ones without having to completely rebuild the system. The key to achieving this is interfaces. When you provide an interface, you can include any component that complies with or offers that interface in the executable component. By allowing components to offer new services through different interfaces that other components can rediscover and utilise, you can grow the system. This semantics describes the purpose behind the UML component definitions. A component is a genuine, replaceable piece of a system that fits and makes a set of interfaces possible.

A component is physically present first. Live in a concept-free universe of bits. A component can be changed, second. One element is exchangeable. A component that fits the same interfaces can be swapped out for another. The technique for adding or removing a component to create a runtime system is typically invisible to the component's user and is activated by object models (like COM + and Enterprise Java Beans), which call for little to no intermediate transformation or processing. automate the system

Third, an element is a component of a system. Rarely does a component exist alone. Instead, each component works together with others to provide the architectural or technological environment in which it is meant to be used.

A component is a crucial structural and/or behavioural portion of a larger system because it is logically and physically coherent. Many different systems can use the same component. Therefore, a component is a fundamental building block that may be used to create and construct systems. Recursion exists in this definition. A component at a higher level of abstraction can easily be a system at a lower level.

The realisation of a collection of interfaces while encapsulating a component, as was previously stated, is the fourth step.

**Component types :**

There are three different kinds of components.

The implementation components come first. These elements, such as dynamic libraries (DLLs) and executable files, are both essential and sufficient to create an executable system (EXE). The definition of a UML component is sufficiently inclusive to cover both traditional object models like COM +, CORBA, and Enterprise Java Beans as well as alternative object models that might include dynamic web pages, database tables, and executable files with proprietary communication protocols.

Second, the work product contains components. These components, which include things like source code files and data files from which the implementation components are constructed, are essentially the leftovers of the development process. Although these parts are the development work products utilised to make the executable system, they do not directly take part in an executable system.

The elements of execution come in third. These parts are produced by a system that is active, such a COM + object that is instantiated from a DLL.

**Organizing Components :**

Similar to how you organise classes, components can be organised by being grouped together in packages.

Components can also be arranged in a hierarchy by having their dependencies, generalisations, associations (including aggregation), and realisations specified.

**Standard Elements :**

Components can use any extension mechanism supported by UML. The majority of the time, you will utilise stereotypes to define new component types and tagged values to extend component characteristics (such as declaring the version of a development component) (such as OS-specific components).

That UML defines five standard stereotypes to apply –one components:

**Table 13.1 : standard stereotypes**

1. executable	Specifies a component there can be executed on a node
2. library	Specific static or dynamics object library
3. table	Specific component that represents a database table
4. file	Specifies a component that represents a document containing source code or data
5. document	Specifies a component that represents a document

❑ **Check Your Progress – 1 :**

1. In component diagram, the interfaces are linked using \_\_\_\_\_  
a. Interfaces    b. connectors    c. components    d. None of these
2. Component diagram is a more specialized form of \_\_\_\_\_ diagram.  
a. class    b. usecase    c. sequence    d. None of these

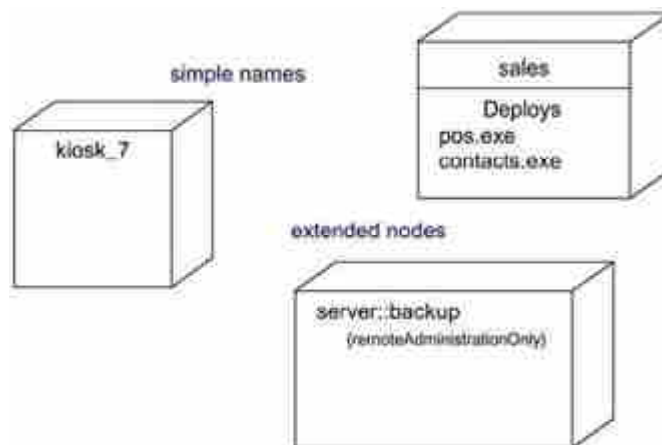
**13.3 Deployment :**

**Terms and Concepts :**

A node is a physical component that persists when a computer is functioning. It stands for a computational resource and typically has memory and processing capacity. A node appears graphically as a cube.

**Names :**

Each node needs a name that sets it apart from the others. A name is a string of text. A path name is the name of the node followed by the name of the packet in which that node lives; a simple name is all that is known about that name. In most cases, a node is represented with nothing except its name visible, like in the figure. You can draw nodes with highlighted values or extra space to expose their subtleties, much like in classes.



**Figure 13.4 : Simple and Extended Nodes**

**Note :**

A node name is a string of text that can span numerous lines and contain any combination of letters, numbers, and punctuation—with the exception of columns, which are intended to distinguish between a node's name and the name of the packet it is contained in. Short nouns or nouns derived from the implementation vocabulary are typically used as node names in practise.

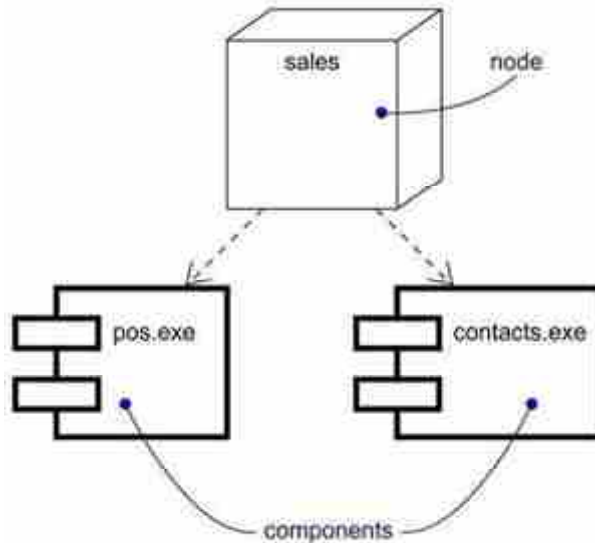
**Nodes and Components :**

Nodes and components are quite similar in many ways: both have names; they can engage in dependencies, generalisations, and associations; both can be embedded; both may have occurrences; and both can take part in interactions. But there are several notable distinctions between nodes and components.

- ✓ Components and nodes work together to execute components, which are the things that contribute to a system's performance.
- ✓ Nodes indicate the actual insertion of components, while components themselves represent the physical packing of the opposing logical objects.

The most significant distinction is the first one. In a nutshell, nodes execute components, which are actions performed by nodes.

The second distinction points to a connection between classes, components, and nodes. A node is the location where the components are implemented, and a component is specifically the materialisation of a group of other logical elements, such as classes and collaborations. One or more components may implement a class, and one or more nodes may implement a component. A dependency relationship can be used to explicitly depict the connection between a node and the components it implements, as illustrated in the picture. Most of the time, you can keep track of these relationships as part of the note specification rather than having to graphically represent them.



**Figure 13.5 : Nodes and Components**

A distribution unit is a collection of items or parts that are collectively assigned to a node.

**Note :**

Nodes are just like classes in that you may give them characteristics and operations. For instance, you can define that a node offers memory attributes, on, off, and suspend activities, as well as CPU speed.

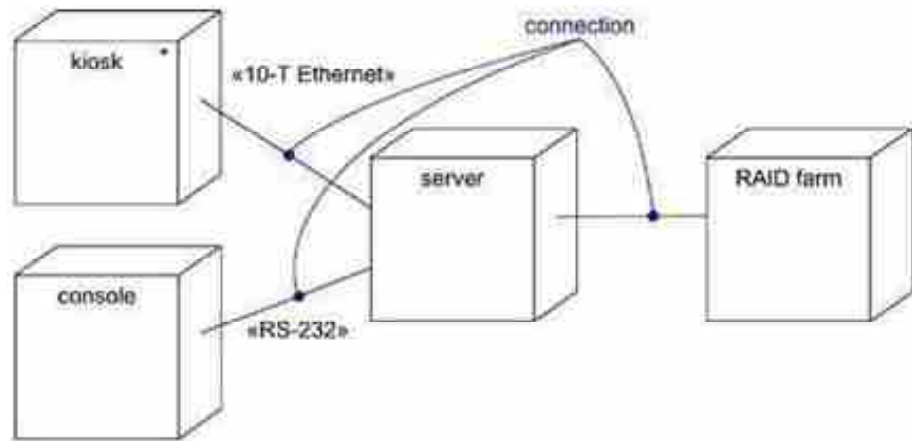
**Organizing Nodes :**

Similar to how you can organise classes and components, you can organise nodes by grouping them in packets. Nodes can also be arranged by defining the dependencies, generalisations, and associations (including aggregation) that exist between them.

**Connections :**

An association is the most typical kind of relationship you want to employ between nodes. As seen in the illustration, an association in this case denotes a physical link between two nodes, such as an Ethernet connection, a serial line, or a shared bus. Associations can even be used to simulate a satellite link between distant processors or other ad hoc indirect connections.





**Figure 13.6 : Connections**

You have access to the complete range of relationships since notes function similarly to classes. This implies that roles, variety, and constraints may all be present. If you want to model additional kinds of connections, you must stereotype these linkages, just like in the preceding picture. To distinguish between an RS-232 serial connection and a 10-T Ethernet connection, for instance.

**❑ Check Your Progress – 2 :**

1. \_\_\_\_\_ are the common notations for deployment diagram ?
  - a. stereotypes
  - b. Artifacts and nodes
  - c. Components
  - d. None of these
2. \_\_\_\_\_ are types of nodes used in the deployment diagram.
  - a. Device & execution environment
  - b. Device
  - c. Execution environment
  - d. Artifact

**13.4 Component Diagrams :**

**Terms and Concepts :**

A component diagram displays a group of components along with their connections. A component diagram is a collection of arcs and vertices on a visual level.

**Common features :**

A component diagram is merely a particular kind of diagram that has the same name and graphic content that is a projection on a model as all other diagrams. A component chart's unique content is what distinguishes it from all other chart styles.

**Contents :**

Component diagrams normally cover

- Components
- Interfaces
- Dependence, generalisation, association, and realization relationships Like all other diagrams, component diagrams may contain notes and constraints.

Packages or subsystems, which are used to aggregate aspects of your model into bigger parts, can also be found in component diagrams. When you wish to see a specific example of a family of component-based systems, you may also want to include instances in your component diagrams.

**Common Uses :**

Component diagrams are used to represent a system's static implementation view. This view primarily enables configuration control for system components that can be put together in different ways to create a functioning system.

Component diagrams can be used in one of four ways to model the static deployment perspective of a system.

**1. To model source code :**

You write code utilising integrated development environments that keep your source code in files while using the majority of contemporary object-oriented programming languages. Component diagrams can be used to depict the configuration management of these files, which serve as work product components.

**2. To model executable versions :**

A set of artefacts that are sent to an internal or external user as part of a release are generally standard and comprehensive. Component releases concentrate on the elements required to produce a functioning system. The physical components that make up your software, or your deployment components, are visualised, specified, and documented when you model a release using component diagrams.

**3. To model physical databases :**

Consider a physical database as the real-world embodiment of a schema existing in the digital universe. The model of a physical database reflects the storage of this data in the tables of a relational database or on the pages of an object-oriented database; forms really offer an API for persistent data. It represents these and other kinds of physical databases using component diagrams.

**4. To model adaptive systems :**

Some systems are rather static; their parts arrive on the scene, take part in an action, and then go. For load balancing and failover, other systems are more dynamic and contain movable agents or components. Component diagrams and a few UML diagrams are used to model behaviour in order to portray this kind of systems.

**❑ Check Your Progress – 3 :**

1. Component diagram is considered as \_\_\_\_\_ diagram.
  - a. Structured
  - b. Behavioural
  - c. (a) and (b) both
  - d. None of these
2. \_\_\_\_\_ diagram describe the organization of source code, binary code and executable.
  - a. class
  - b. usecase
  - c. sequence
  - d. component

## **13.5 Deployment Diagrams :**

### **Terms and Concepts :**

A deployment diagram is a visual representation of how the components on the run-time processing nodes are configured during operation. An implementation diagram looks as a group of arcs with a central vertex.

### **Common Properties :**

A deployment diagram is merely a particular kind of diagram, and all diagrams have the same basic elements: a name and graphic content that is projected onto a model. The precise content of an implementation chart is what distinguishes it from all other chart kinds.

### **Contents :**

Deployment diagrams generally contain

- Nodes
- Dependence and association relationships

Deployment diagrams, like any diagrams, can include annotations and restrictions.

Components that must all be located on one or more nodes may also be shown in deployment diagrams. Packages or subsystems, which are used to organise pieces of your model into bigger portions, can also be included in implementation diagrams. When you wish to show a specific example of a group of hardware topologies, you may also want to include instances in your deployment diagrams.

### **Note :**

A deployment diagram is really just a specific variety of class diagram that emphasises system nodes.

### **Common uses :**

Deployment diagrams are used to represent a system's static implementation view. The distribution, delivery, and installation of the components that make up the physical system are the main topics covered by this viewpoint.

For certain kinds of systems, deployment diagrams are not necessary. You can disregard deployment diagrams on It if you create software that runs locally on a device and only communicates with common devices on the device that are already controlled by the host operating system (such a keyboard, monitor, and modem for a personal computer). The use of deployment diagrams, on the other hand, will assist you in thinking through the system's software to hardware mapping if you are developing a piece of software that interfaces with devices that are not typically managed by the host operating system or that are physically distributed across multiple processors.

You will often utilise deployment diagrams in one of three ways when modelling the static deployment perspective of a system.

#### **1. To model embedded systems :**

An embedded system is a complex group of hardware that communicates with the outside world through software. Software is used in embedded systems to control components like motors, actuators, and monitors. These components are in turn controlled by external stimuli including sensor input, motion, and

temperature changes. The components of an embedded system, such as the devices and CPUs, can be modelled using deployment diagrams.

**2. To model client / server systems :**

A typical architecture that emphasises creating a distinct division of interests between the system's user interface (which is located on the client) and the system's permanent data is known as a client/server system (residing on the server). In client/server systems, which represent one end of the distributed system continuum, you must decide how to link clients to servers through a network and how to physically distribute the software components of your system among nodes. Implementation diagrams can be used to model the topology of such systems.

**3. To model fully distributed systems :**

Systems that are widely distributed, if not globally distributed, and often involve numerous levels of servers, are at the other extreme of the spectrum of distributed systems. Multiple versions of software components are frequently hosted by such systems, and some of them may even move between different nodes. Making choices throughout the construction of such systems must allow for constant topological change.

Deployment diagrams help you see the present system topology and component layout so you can analyse the effects of topology changes.

**❑ Check Your Progress – 4 :**

1. Which of the following is correct ?
  - a. Artifacts names and instances are underline
  - b. Artifacts instances and types have same names
  - c. Both a and b
  - d. None of these

**13.6 Let Us Sum Up :**

In this unit, we have learned that an architecture model is a partial abstraction of a system. It is an approximation, and it captures the different properties of the system. We have also seen that architecture modelling involves identifying the characteristics of the system and expressing it as models so that the system can be understood.

Components are physical, replaceable system parts that comply to a set of interface specifications and enable the realisation of those interfaces. We also discovered that a component diagram displays a group of components along with their connections. And a deployment diagram is a diagram that displays the setup of the components housed on run-time processing nodes.

**13.7 Answers for Check Your Progress :**

- ❑ Check Your Progress 1 :**  
 1 : b            2 : a
- ❑ Check Your Progress 2 :**  
 1 : b            2 : a

❑ **Check Your Progress 3 :**

1 : a            2 : d

❑ **Check Your Progress 4 :**

1 : b

**13.8 Glossary :**

1. **Name** – Each component must be identified by a name that sets it apart from other components.
2. **Component** – It is a genuine, replaceable component of a system that fits and makes a set of interfaces possible.
3. **Node** – A node is a physical component that exists at runtime and represents a computing resource. Nodes typically have processing power and at least some memory. A node is represented graphically as a cube.

**13.9 Assignment :**

1. Explain component in detail.

**13.10 Activities :**

1. Discuss deployment in detail.

**13.11 Case Study :**

1. Discuss process to develop the component diagram and deployment diagram.

**13.12 Further Readings :**

1. Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young – Object-Oriented Analysis and Design with Applications
2. Chris Raistrick, Paul Francis, John Wright, Colin Carter– Model Driven Architecture with Executable UML

**UNIT STRUCTURE**

- 14.0 Learning Objective
- 14.1 Introduction
- 14.2 UML Diagrams : Library Management System
- 14.3 Online Mobile Recharge
- 14.4 Let Us Sum Up
- 14.5 Answers for Check Your Progress
- 14.6 Glossary
- 14.7 Assignment
- 14.8 Activities
- 14.9 Case Study
- 14.10 Further Readings

**14.0 Learning Objectives :**

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

- About the UML diagram for two systems
- About Library Management System
- About the online mobile recharge system

**14.1 Introduction :**

UML is a graphical modelling language that supposedly unites and integrates the various notations previously used by the various proposed methods. This is the requirement as the number of object-oriented methods increased from less to much more. It constitutes the standard notation and semantics necessary to accurately describe software created with object-oriented or component-based technology. It is certainly a step in the right direction; however, it is not an ideal or universal modelling language. We tend to think that the UML, as it is seen today, needs to be supplemented, in some contexts and for some application domains, by alternative metamodels or at least adapted to address these metamodels.

➤ **UML Diagrams – Library Management System :**

**Library Management System :**

Library Management System Case Study is a library management software that is used to monitor and control the transactions found in the library. The case study provides a detailed description of the library and daily transactions. To keep track of new and retrieve details about books found in the library, details about basic actions such as :

- add new member
- new books

**Object Oriented  
Analysis and Design**

- search for books
- action of the partners in relation to the loans and returns of books

The case study offers attractive, familiar and well-thought-out user interfaces, combining search, deployment, and reporting details. The study focuses on the reporting facility in the library system that helps to get a good idea about the members. The description shows functions obtained using the case study:

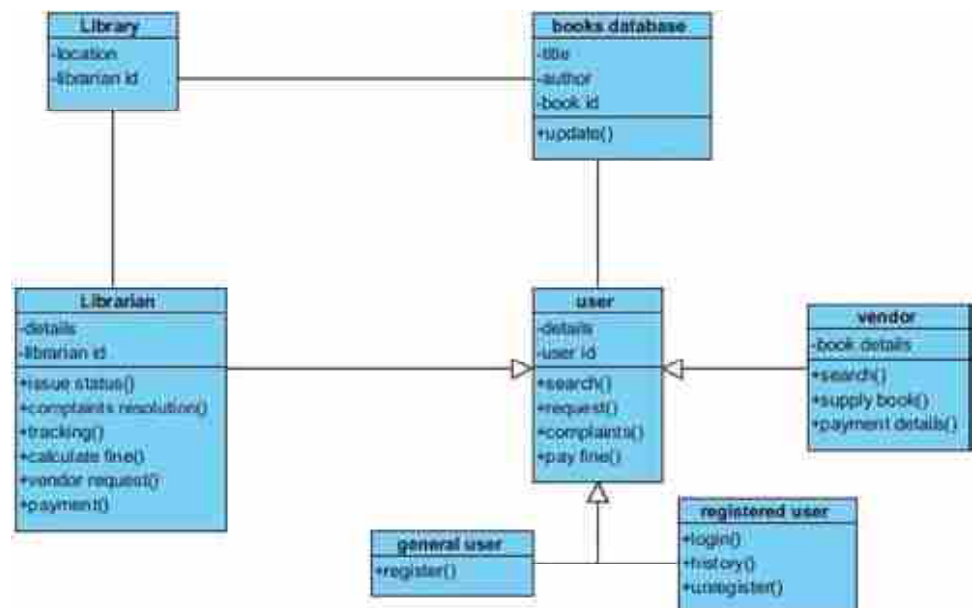
**End users :**

- Librarian: To maintain and update the records and also attend to the needs of the patrons.
- Reader: needs books to read and also makes several inquiries to the librarian
- Vender: Supply and comply with the requirements of the prescribed books.

**Class Diagram :**

**Identified classes :**

- Library
- Librarian
- Book Database
- User
- Vendor



**Figure 14.1 : Class Identification**

➤ **Use the case diagram :**

**Actors against use cases:**

**Librarian**

- Issue a book
- Update and maintain records
- Request the vendor for a book
- Track complaints

**User :**

- Register
- Login
- Search a book
- Request for issue
- View history
- Request to the Librarian
- Unregistered

**Books Database :**

- update records
- State of the book

**Vendors :**

- Provide books to the library
- Payment acknowledgement

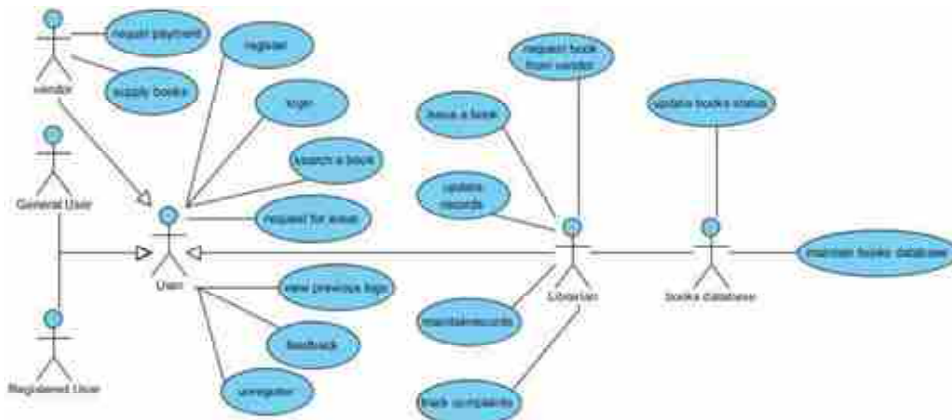


Figure 14.2 : Use Case Diagrams

**Sequence Diagram :**

Figure 14.3 shows a sequence diagram that describes the process of searching for books to issue books as requested by the user to the librarian :

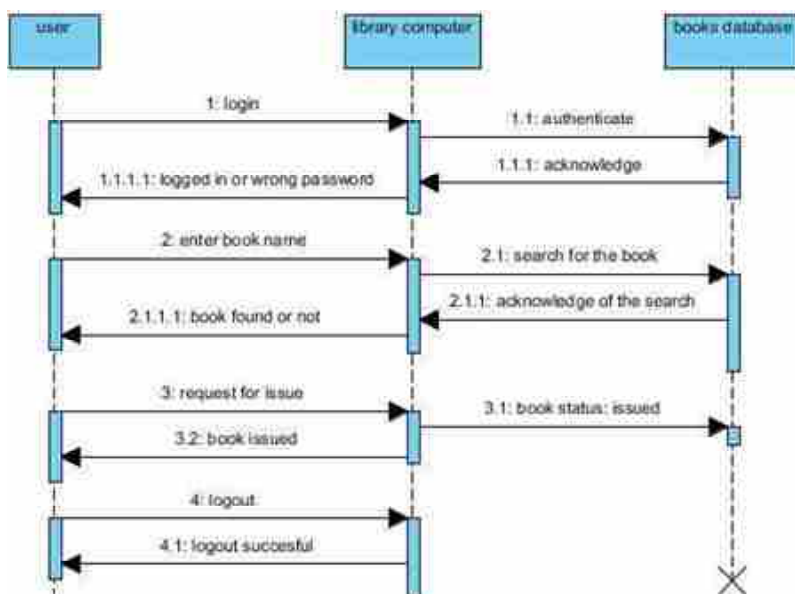
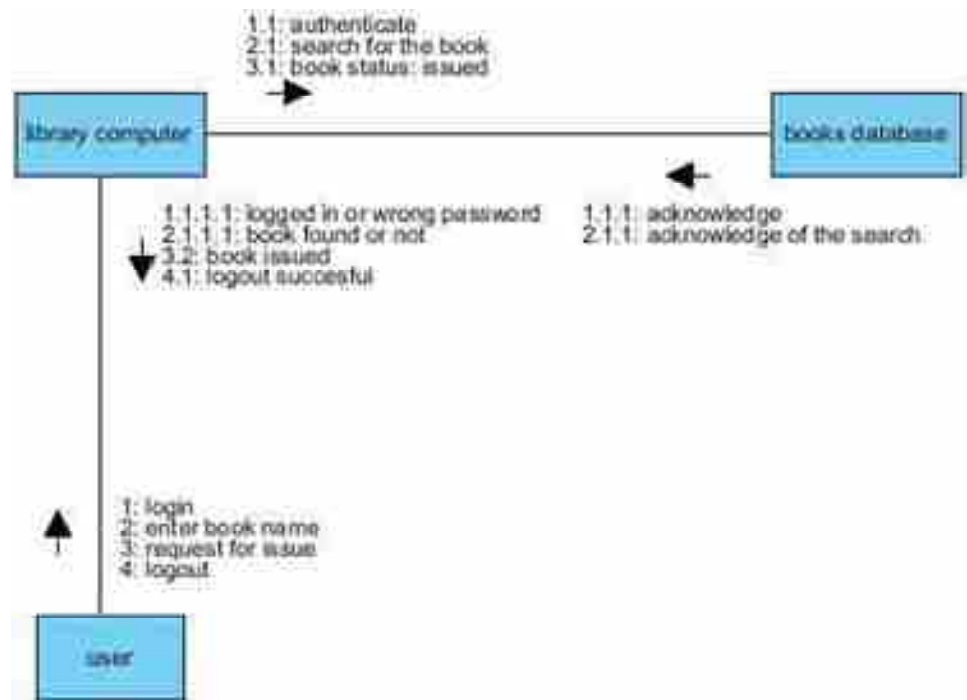


Figure 14.3 : Sequence Diagram



➤ **Collaboration Diagram :**

Figure 14.4 shows an overview of the collaboration diagram, which is used to position the book and show the issuance process as required by the librarian user:

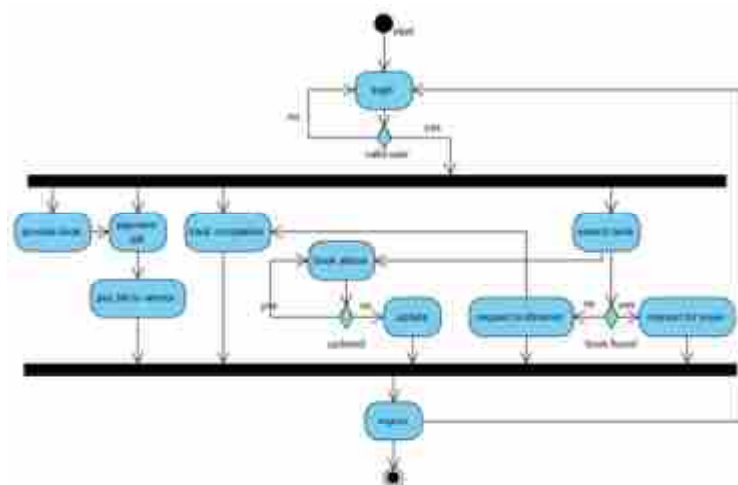


**Figure 14.4 : Collaboration Diagram**

➤ **Activity Diagram :**

**Activities :**

- User Login and Authentication
- Search book operation for Reader
- Acknowledge and Issue books to the users by the Librarian
- Provide books requested by the Librarian from the Vendor
- Bill payment from the Librarian to the Vendor
- Status of the books updated in the Books Database



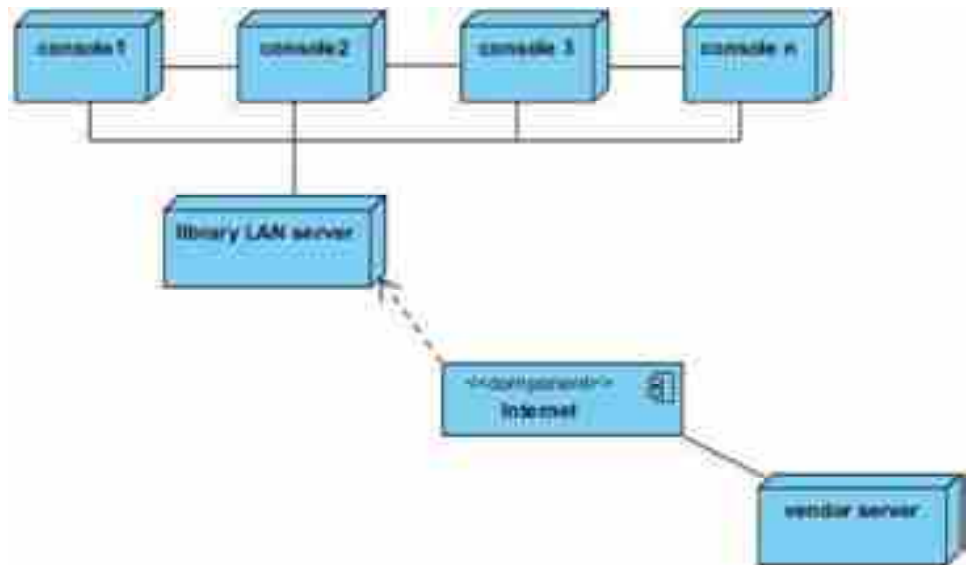
**Figure 14.5 : Activity Diagram**



➤ **Deployment Diagram :**

**Systems used :**

- Local Consoles / Computers for login and search purposes by users, librarian and vendors.
- Library LAN Server interconnecting all the systems to the Database.
- Internet to provide access to Vendors to supply the requested books by the Librarian.
- Vendor Server to maintain the records of the requests made by the librarian and books provided to the library.



**Figure 14.8 : Deployment Diagram**

❑ **Check Your Progress – 1 :**

1. What are the benefits of using object modelling ?
  - a. Model the objects of the problem domain.
  - b. You can easily map the records to the corresponding database table
  - c. It is used as a return value for data access object methods.
  - d. All these
2. Interactions between messages are classified by which diagram ?
  - a. State chart    b. activity    c. collaboration    d. Object
3. Which diagram from the following is help to show dynamic aspects related to system ?
  - a. State chart    b. activity    c. collaboration    d. interaction

**14.3 Online Mobile Recharge :**

This case study involves mobile online recharge, which provides detailed information on the activities of mobile service providers. It shows how such an application details information about the mobile service provider, which can be in the areas of :

- Plans
- Options
- Advantage

The study provides information related to various schemes and services that the company provides to its customers, where you can choose any plan, recharge option, billing date and details regarding any mobile recharge application. The benefit of said study makes it easier for the client to have recharge facilities with any service provider on the same platform.

- End users:
- Service Provider:

Service provider is any company that provides mobile services to customers related to mobile connections along with applications related to mobile charging and multiple plans. The request comes from clients who are further verified on the administrator side who help check the balance and clients request certain mobile services and recharge the application.

**Third Party System Administrator :**

An administrator is a person who oversees each mobile user and their transactions and manages service providers in terms of user account management. When inserting a user, the administrator will verify the available balance in the user's account and will subsequently request the service provider to process the information. Admin carries all the information about the user related to recharge schemes and options.

**User :**

There are 2 categories in the user module :

- Registered user
- Visitor

Any customer who needs to use the Online Mobile Recharge services at any given time and anywhere must register and then be able to recharge their mobile online or from any counter. Visitors are those who visit the online mobile recharge application and after completing the required information about the service providers, they will be able to recharge the mobile online after entering all the bank details or debit/credit card details.

➤ **Class Diagram :**

**Identified classes:**

- User : Registered, Visitors
- Third party system administrator
- Third party server/database
- Service provider
- Direct or non-third-party user

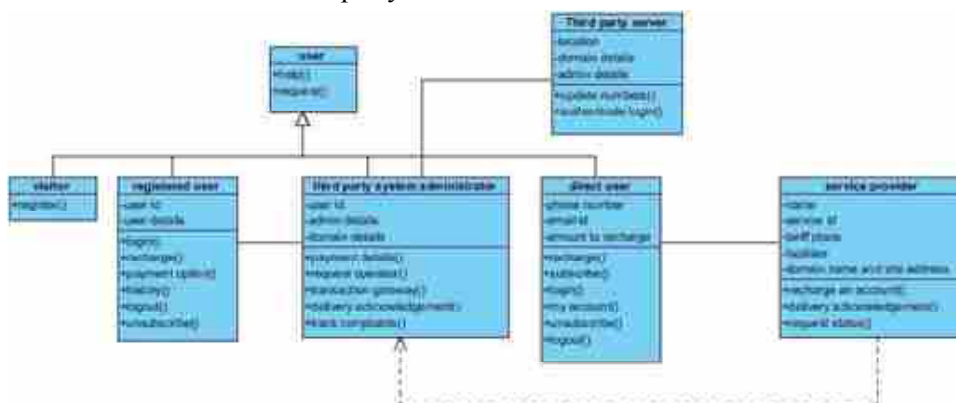


Figure 14.9 : Class Diagram

**Object Oriented  
Analysis and Design**

➤ **Use Case Diagram :**

**Actors Vs Use Cases :**

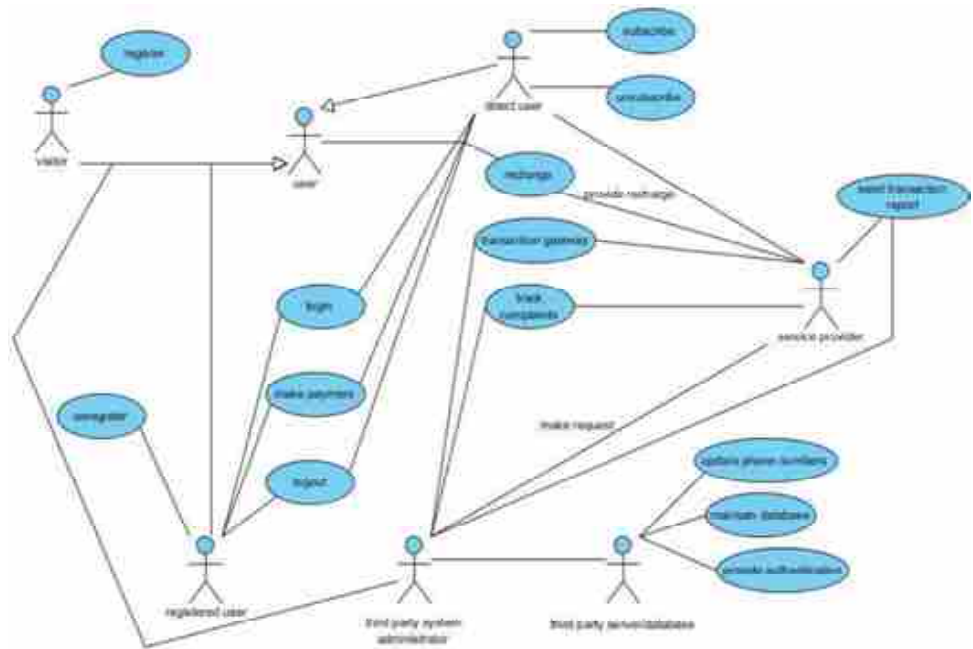
**User :**

- Register.
- Recharge.
- Select Payment Gateway.
- Select Service Provider.
- Make payment.
- Third Party Administrator
- Forward User request to Service Provider
- Track Complaints.

**Third party server/database :**

**Service Provider :**

- Recharge the user requested either directly or through the third party system.
- Provide various plans to the user.



**Figure 14.10 : Use-Case Diagrams**

Third party server/database

Service Provider

➤ **Sequence Diagram :**

Figure 14.11 shows the layout of the mobile recharge system sequence diagram, where the user can recharge his account using a third party :

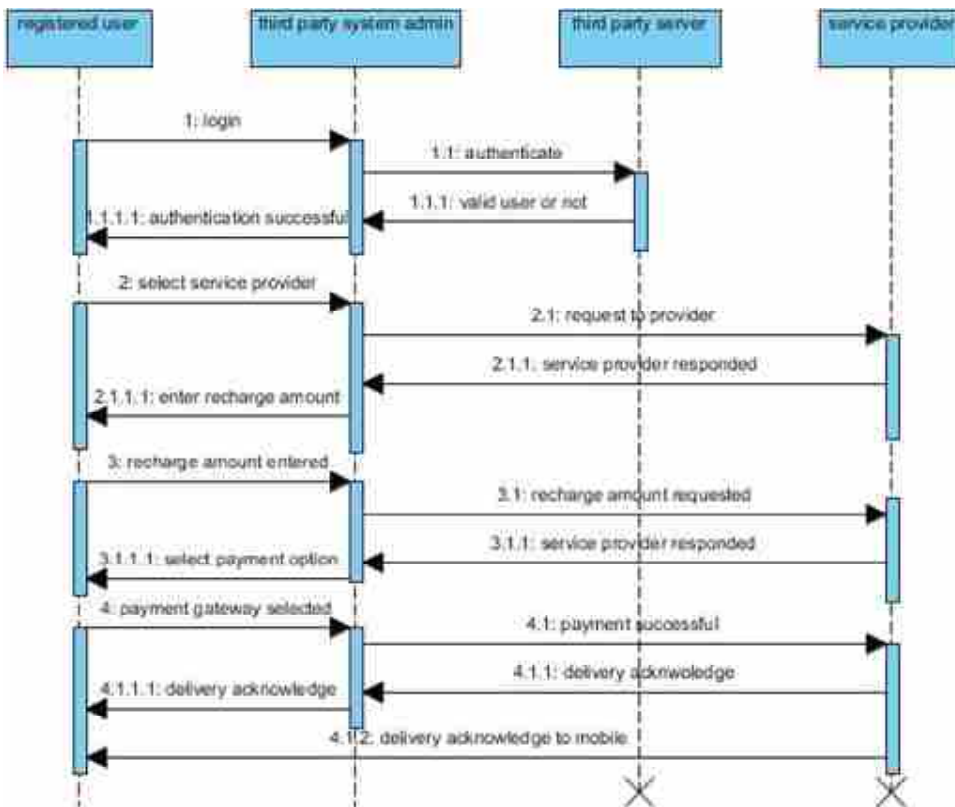


Figure 14.11 : Sequence Diagram

➤ Collaboration Diagram :

Figure 14.12 shows the details of the collaboration diagram that describes the mobile top-up of users who top up their account through the third-party site :

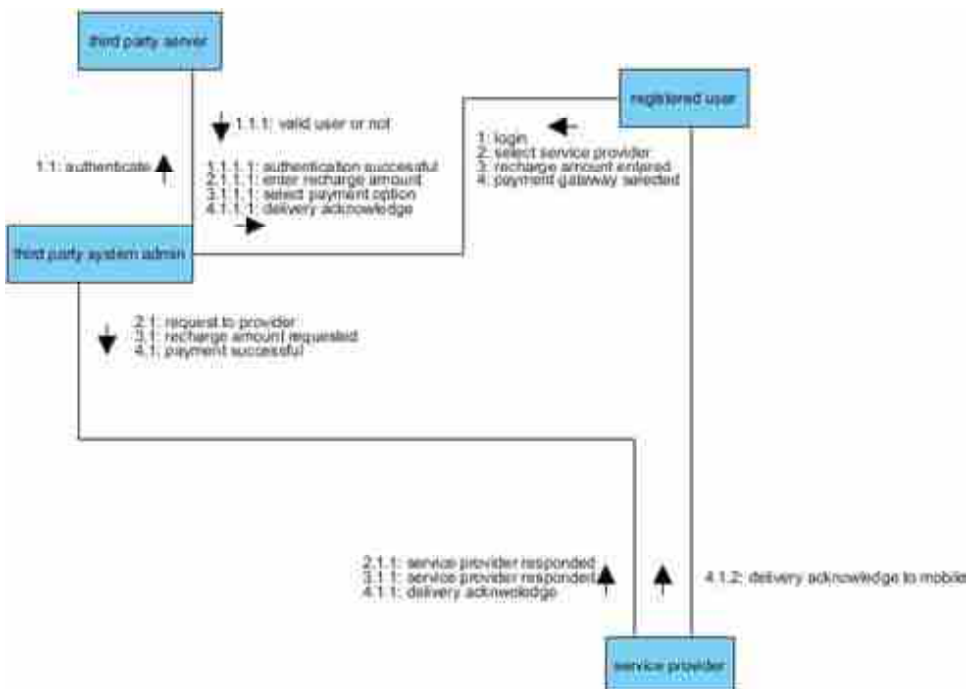


Figure 14.12 : Collaboration Diagram

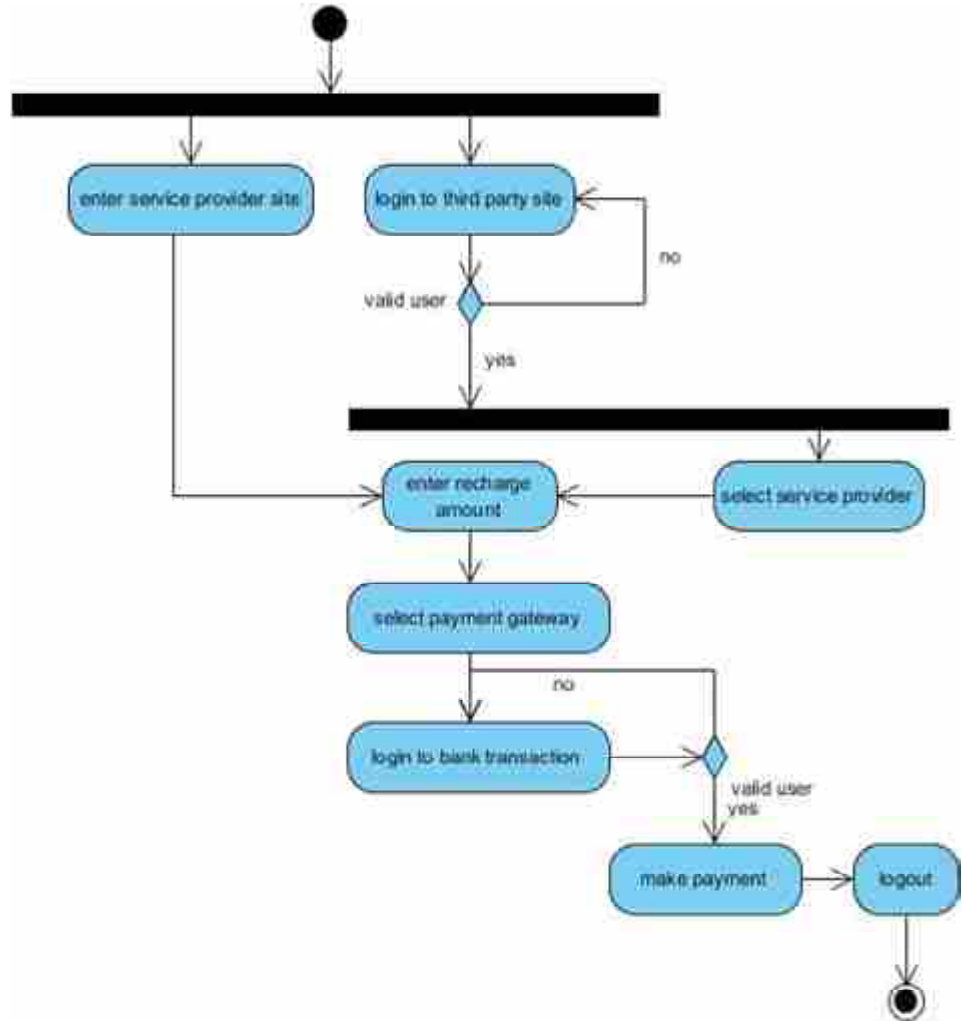
➤ Activity Diagram :

Activities :

- User login and authentication for Registered user.

**Object Oriented  
Analysis and Design**

- Forward the request to service provider if logged in as a Administrator.
- Enter service provider site for a direct user.
- Enter recharge amount.
- Select Payment Gateway.
- Login and authenticate Bank Account.
- Make payment.
- Check for the recharge processed successfully or not.



**Figure 14.13 : Activity Diagram**

■ **State Chart Diagram :**

**State :**

- Authentication for registered users / Registration for unregistered users
- Successfully logged on or re-login
- Operator Selection
- Show the tariff plans available and applicable
- Request recharge
- Go through Payment Gateway Transaction process
- Authentication to enter the gateway site
- Successfully logged on or re-login
- Payment made

➤ **Logged off :**

**Transitions :**

- Registration --> Authenticate --> Logged in
- Logged in --> Operator Selection --> Tariff Plan <--> Request Recharge
- Operator Selection --> Request Recharge --> Payment Gateway Transaction
- Payment Gateway Transaction --> Operator Selection
- Payment Gateway Transaction --> Logged off

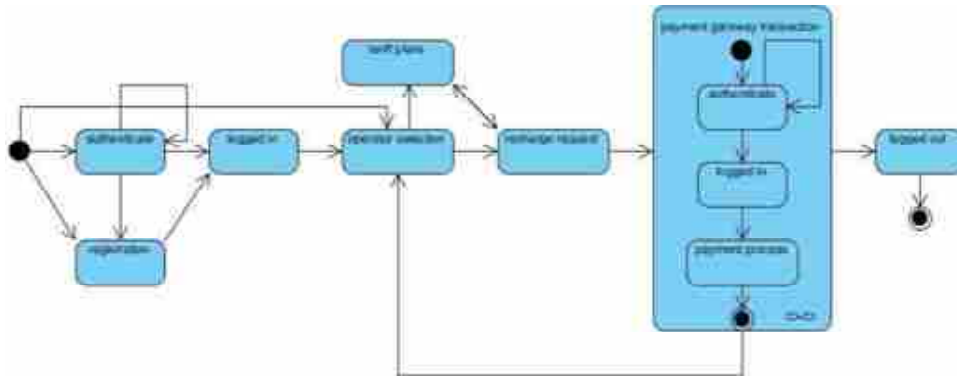


Figure 14.14 : State Chart Diagram

➤ **Component Diagram :**

**Components :**

- Third Party Home Page (visitor / registered user / admin / service provider)
- Third Party Register Page (visitor)
- Third Party Login Page (registered user)
- Third Party User History Page (registered user)
- Request Recharge Page (registered user)
- Third Party Logout Page (registered user)
- Online Payment Transaction Gateway Page (direct user / registered user)
- Service Provider Home Page (visitor / registered user / admin / service provider)
- Tariff Plans Page (visitor / registered user / admin / service provider)

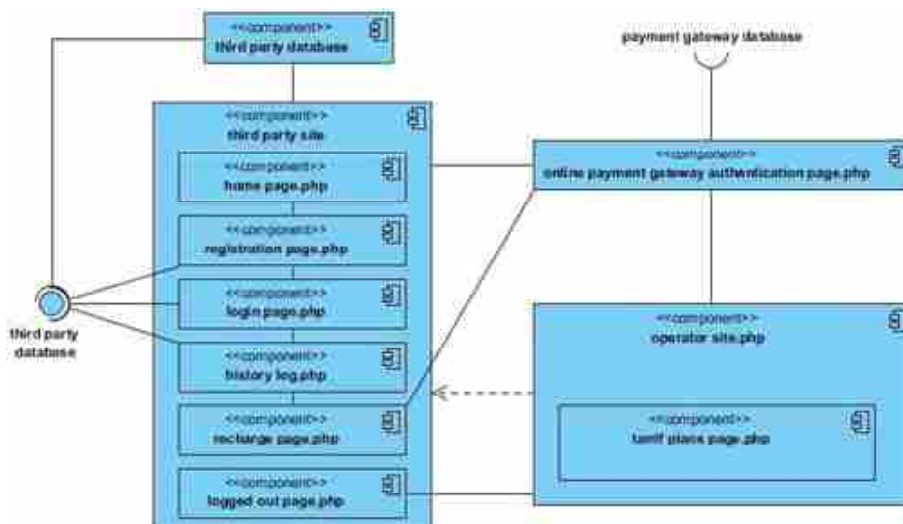


Figure 14.15 : Component Diagram

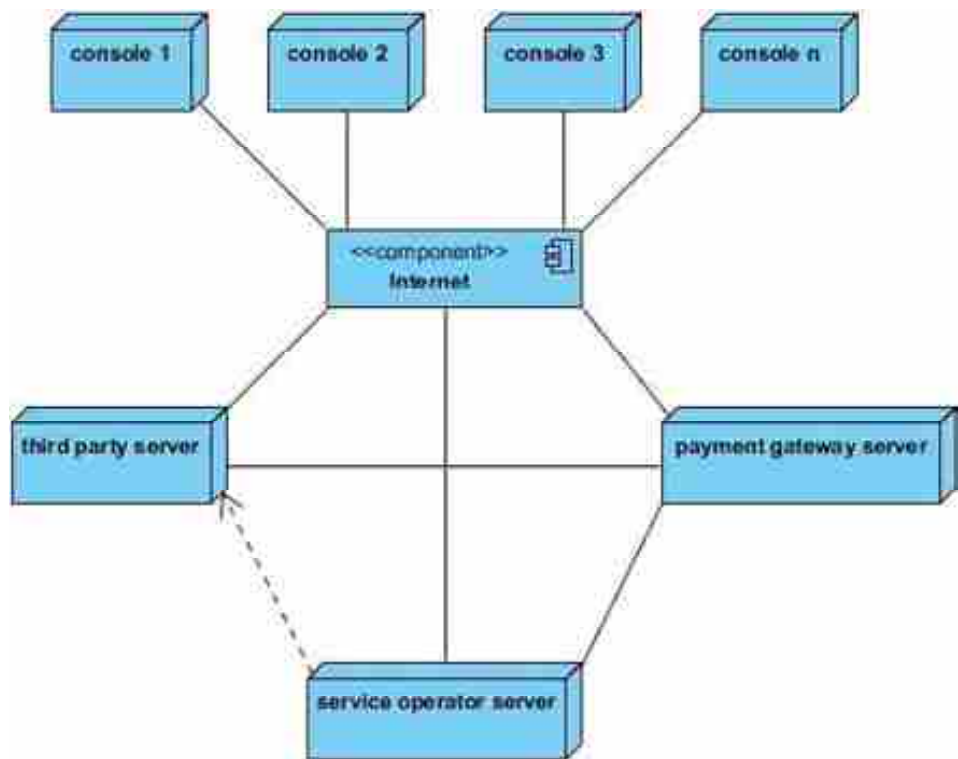


**Object Oriented  
Analysis and Design**

➤ **Deployment Diagram :**

**Systems used :**

- Consoles / Computers for registration, login purposes by third party users and for quick recharge by direct users.
- Third Party Server to receive and respond to all the requests from various users.
- Internet to provide access to users to recharge their accounts through payment gateways by placing requests through Third Party Site and Service Providers sites.
- Payment Gateway Serve like Bank's server to provide online payment through their personal accounts to meet the requirements of the users.
- Service Provider Server to maintain the records of the requests made by the users.



**Figure 14.16 : V**

❑ **Check Your Progress – 2 :**

1. Which of the following diagrams is used to show the hardware and software of the system ?
  - a. class diagram
  - b. Implementation diagram
  - c. package diagram
  - d. None of those
2. Class consists abstractions from the following. Select from the following.
  - a. Operations
  - b. Objects
  - c. Attributes
  - d. All of these
3. Data item is held by \_\_\_\_\_ from the following.
  - a. Class
  - b. Object
  - c. None of these
  - d. All of these

#### 14.4 Let Us Sum Up :

In this unit, we have learned that UML contains 2 different metamodels such as state diagrams and activity diagrams with different features of reactive systems, especially concurrency and hierarchy.

A class diagram, which serves as the foundation of object-oriented response and displays classes in higher systems, attributes, and class operations, is likely the most used sort of UML diagram.

Instance diagrams, which are comparable to class diagrams in that they show the relationships between objects used in the real world and are used to represent the system's appearance at a particular time, are called object diagrams.

When a system's behaviour is modelled using activity diagrams, it signifies that various behaviours are interconnected and form the system's overall flow.

#### 14.5 Answers for Check Your Progress :

**Check Your Progress 1 :**

1 : d          2 : c          3 : d

**Check Your Progress 2 :**

1 : b          2 : d          3 : d

#### 14.6 Glossary :

1. **UML** : a kind of metamodel with different properties for reagents systems
2. **Class Chart** : An arguably prominent UML chart type that serves as building block of the object-oriented response that shows the classes in the system, the attributes, and the class operations.
3. **Object Diagram** : similar to class diagrams that show the relationships between real-world objects that describe the appearance of the system at a given time.
4. **Activity Diagram** : The model that shows the behavior of the system, where the behavior is related to the general flow of the system.

#### 14.7 Assignment :

1. Write a short note about UML diagrams.

#### 14.8 Activities :

1. Gather information about mobile recharge online.

#### 14.9 Case Study :

1. Formulation generalized problems statement and discuss it.

#### 14.10 Further Readings :

1. Schulte, C., & Niere, J. (2002, June). Thinking in object structures: Teaching modelling in secondary schools. In Sixth Workshop on Pedagogies and Tools for Learning Object Oriented Concepts, ECOOP, Malaga, Spanien.
2. Zündorf, A. (2001). Rigorous object oriented software development (Doctoral dissertation, Habilitation Thesis, University of Paderborn).

**BLOCK SUMMARY :**

In this block, we learned what the unified modelling language is and the importance of modelling for application development. We also discussed the conceptual model of UML and the detailed explanation of its element such as things (object-oriented parts of UML), reporters and diagrams used in UML. In this block we learnt that Behavioural Modelling, purpose of the use cases, Activity diagram which shows the order of activities in a process, including sequential and parallel activities, and the decisions made.

In this block you also learned that an event, signal, state machine diagram. You have learned that an architecture model, component, component diagram, component and their relationship, also the deployment diagram. We have learned about component which is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces. We also learned that a component diagram shows a set of components and their relationships. And a deployment diagram is a diagram that shows the configuration of run time processing nodes and the components that live on them. In the last unit of this block, you learnt about the UML diagrams which is required to draw for two different system.

## **BLOCK ASSIGNMENT :**

### ❖ **Short Questions :**

1. What is behavioural modelling ?
2. What are four basic component of behavioural modelling ?
3. What are data modelling techniques ?
4. What are process & threads ?

### ❖ **Long Questions :**

1. Explain use case diagram with example.
2. Explain state chart diagram.
3. Explain component diagram with example.

**Object Oriented  
Analysis and Design**

❖ **Enrolment No. :**

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

Unit No.	11	12	13	14
No. of Hrs.				

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

Items	Excellent	Very Good	Good	Poor	Give specific example if any
Presentation Quality	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Language and Style	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Illustration used (Diagram, tables etc)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Conceptual Clarity	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Check your progress Quest	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
Feed back to CYP Question	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____

3. Any other Comments

.....

.....

.....

.....

.....

.....

.....

.....



# DR.BABASAHEB AMBEDKAR OPEN UNIVERSITY

'Jyotirmay' Parisar,  
Sarkhej-Gandhinagar Highway, Chharodi, Ahmedabad-382 481.  
Website : [www.baou.edu.in](http://www.baou.edu.in)