1... Introduction and Basics of Software Modelling

Learning Objectives ...

- To know about system development life cycle models.
- To know what is system concept.
- To get knowledge about project organization.
- To learn about the communication in project management.
- To know about risks in project management.

1.1 INTRODUCTION

- A system development methodology is an orderly and integrated collection of various methods, tools and techniques.
- There are many approaches to the development of computer system such as:
  1. **System Development Life Cycle (SDLC)**: SDLC is traditional approach which is rigid. It concentrates more on physical aspects than logical aspects for the system development and has to be done at the end of the project.
  2. **Structured System Analysis and Design Method (SSADM)**: SSADM presents graphic model of the system. This methodology involves DFD that represents data movements, data stores and processes of the system. This methodology uses tools like DD, structured English, decision table, decision tree for system analysis.

1.2 SOFTWARE LIFE CYCLE MODELS (REVISION OF SE)

- A software life cycle model is a particular abstraction that represents a software life cycle. A software life cycle model is often called a Software Development Life Cycle (SDLC).
- SDLC concentrates on feasibility analysis, cost benefit analysis, project management hardware and software selection and personnel consideration.

1.2.1 Phases within SDLC

- There are different phases within SDLC, and each phase has its various activities. It makes the development team able to design, create, and deliver a high-quality product.
- The steps/phases or activities involved in SDLC are explained below:

Phase I: System Analysis:

- Primary Investigation or Feasibility Study.
- Analysis or Requirement gathering.

Phase II: System Design:

- Design
- System Implementation and Coding
- System Testing
- Deployment
- Maintenance

1.2.2 Types of SDLC Process Model

- A Software Life Cycle Model is either a descriptive or prescriptive characterization of how software is or should be developed.

1. **Descriptive model:**
   - A descriptive model describes the history of how a particular software system was developed. A descriptive process model is defined as, "a model that describes 'what to do' according to a certain software process system".
   - Descriptive models may be used as the basis for understanding and improving software development process or for building empirically grounded prescriptive models.
2. **Prescriptive model:**
   - Prescriptive models are used as guidelines or frameworks to organize and structure how software development activities should be performed, and in which order.
   - Prescriptive model describes what should be done during software development, including responses to error situations.

Types of Prescriptive model:

- There are three types of prescriptive process models as following:

(i) Waterfall Model:

- In waterfall model, a workflow is in a linear (sequential) fashion. The waterfall model often used with well-defined adaptations or enhancements to current software. In a Waterfall model, each phase must be completed fully before the next phase can begin.

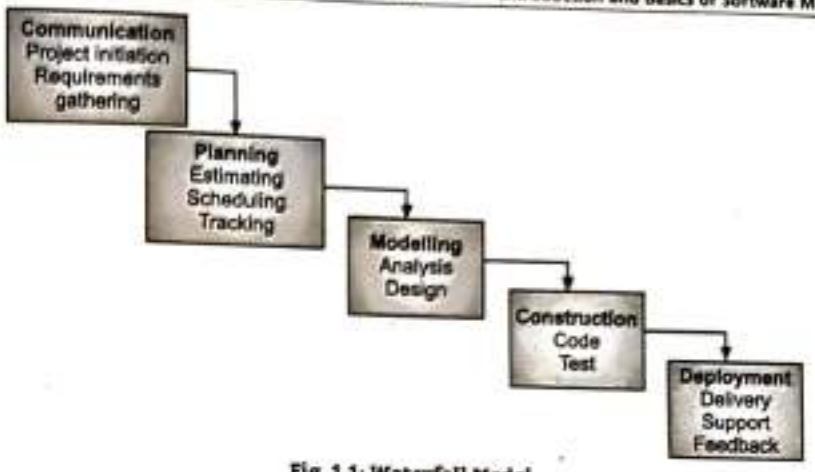


Fig. 1.1: Waterfall Model

Advantages:

1. Waterfall model is a linear model and of course, linear models are the most simple to be implemented.
2. The amount of resources required to implement this model is very minimal.
3. One great advantage of the waterfall model is that documentation is produced at every stage of the waterfall model development. This makes the understanding of the product designing procedure simpler.
4. After every major stage of software coding, testing is done to check the correct running of the code.
5. In waterfall model progress of system is measurable.

Disadvantages:

1. In waterfall model, there is the difficulty of accepting change after the process is in progress. It does not support iteration, so changes can cause confusion.
2. It has high risks and uncertainty.
3. It requires customer patience because a working version of the program does not occur until the final phase.
4. It is a poor model for long and ongoing projects.

(ii) Incremental Process Model:

- The incremental model is defined as, "a model of software development where the product is designed, implemented and tested incrementally (a little more is added each time) until the product is finished."
- In incremental model, multiple development cycles take place that making the software life cycle a multi-waterfall. In this model, cycles are divided up into smaller, more easily managed modules.
- The work flow is in a linear (sequential) fashion within an increment and is staggered between increments. Iterative nature of this model focuses on an operational product with each increment.

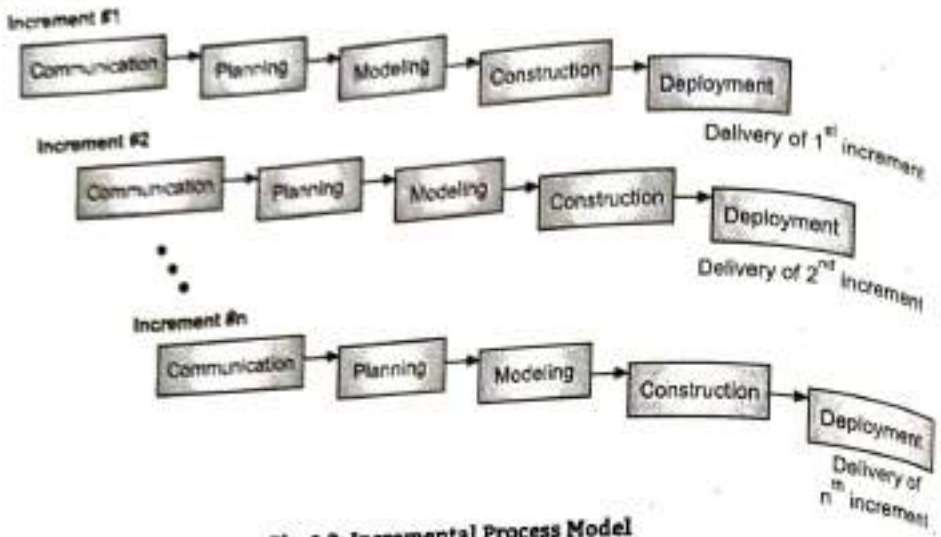


Fig. 1.2: Incremental Process Model

Advantages:

1. This model generates working software quickly and early during the software life cycle.
2. This model is more flexible to change scope and requirements.
3. It is easier to test and debug during a smaller iteration.
4. In this model, customers can respond to each built.
5. Lower initial delivery cost.
6. Easier to manage risk because risky pieces are identified and handled during iteration.

Disadvantages:

1. Needs good planning and design.
2. Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
3. Total cost is higher than waterfall model.

(iii) Evolutionary Process Model:

- Evolutionary model is useful for projects using new technology that is not well understood. This is also used for complex projects where all functionality must be delivered at one time, but the requirements are unstable or not well understood at the beginning.
- It has basic two models i.e. Prototyping and Spiral Model.

(a) Prototyping model:

- Prototyping follows an evolutionary and iterative approach. It provides a working model to the user early in the process, enabling early assessment and increasing user's confidence. Prototyping is used when requirements are not well understood.

- It focuses on those aspects of the software that are visible to the customer/user. User feedback is used to refine the prototype.

- **Phases of Prototyping Model:** The following are the primary phases involved in the development cycle of any prototype model:
  - Requirement gathering
  - Quick Design
  - Building Prototype
  - User Evaluation
  - Refining prototype
  - Engineer product

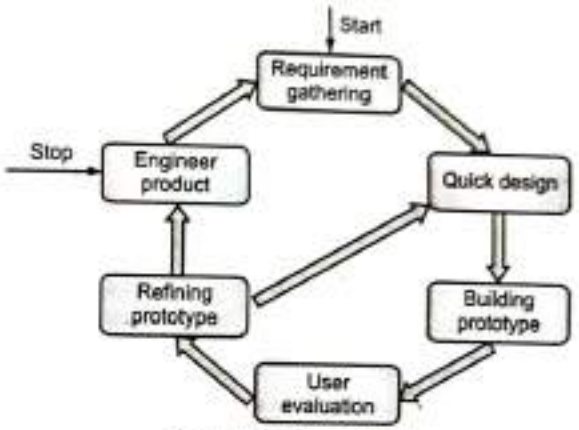


Fig. 1.3: Prototype Model

Advantages:

1. Prototype model need not know the detailed input, output, processes, adaptability of operating system and full machine interaction.
2. In development process of this model users are actively involved.
3. The development process is the best platform to understand the system by the user.
4. Errors are detected much earlier.
5. It gives quick user feedback for better solutions.
6. In this model, missing functionality can be easily identified.
7. It also identifies confusing or difficult functions.

Disadvantages:

1. The client involvement is more and it is not always considered by the developer.
2. It is a slow process because it takes more time for development.
3. Many changes can disturb the rhythm of the development team.

- IEEE defines, the spir... (proces...
which the constituent activities, typical requ... nts analysis, preliminary ...
detailed design, coding, integration and testing are performed iteratively ...
the software is complete".
- This is also called as Process Model. The spiral model of software process ...
originally proposed by Boehm.
- Spiral Model is a combination of a waterfall model and iterative model. Eac...
phase in the spiral model begins with a design goal and ends with the cli...
reviewing the progress.
- Inner spirals focus on identifying software requirements and project risks; ma...
also incorporate prototyping.
- Outer spirals take on a classical waterfall approach after requirements have be...
defined, but permit iterative growth of the software.
- The baseline Spiral starts in the Communication phase, moves in the Planning
phase. In this phase requirements are gathered and risk is assessed. Then it move...
in the Modeling phase, and then in the Deployment phase. Accordingly, eac...
subsequent spiral builds on the baseline spiral.

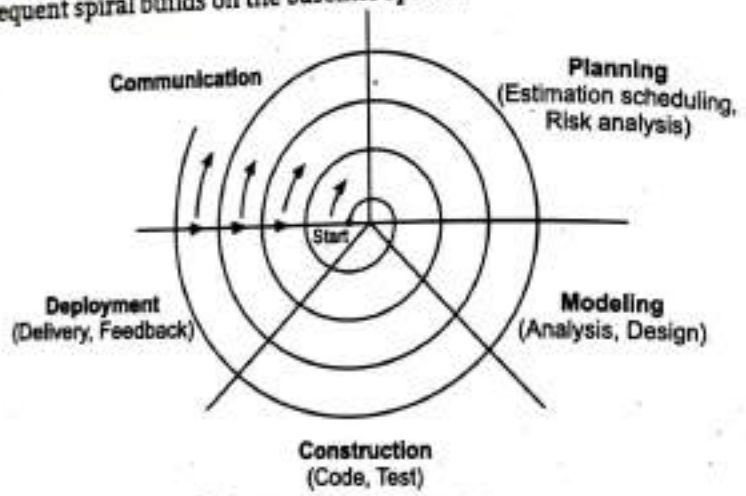


Fig. 1.4: New Spiral Model

Advantages:

1. The spiral model accommodates life-cycle evaluation, growth and requirement changes.
2. It focuses on early error detection and design flaws.
3. It incorporates prototyping as a risk-reduction strategy.
4. The spiral model incorporates software quality objectives into the product.

Disadvantages:

1. Spiral model can be a costly model to use.
2. It is not suitable for low risk projects.
3. Risk analysis in this model requires highly specific expertise.

1.3 SYSTEM CONCEPTS

- In this section, we will see about definition of system, its components, elements and types of system.

1.3.1 Definition of System

- The word system is derived from the Greek word "systema" which means the organized relationship among the functioning units.
- A system is an orderly grouping of interdependent components linked together according to a plan to achieve a specific goal/objective.
- A system is defined as, "a collection of related components that interact to perform a task in order to accomplish a goal".

OR

- A system can be defined as, "an integrated collection of components which satisfy functions (tasks) necessary to achieve the system's objectives or goals".

1.3.2 Characteristics of a System

- Following are some important characteristics of a system:

1. **Organization:**
   - Organization implies structure and order.
   - It is the arrangement of components that helps to achieve pre-determined objectives/goals.
   - For example, a computer system is designed around an input device, a central processing unit and an output device with one or more storage units; when linked together they work as a whole system for producing information.
2. **Interaction:**
   - It is defined by the manner in which the components operate with each other.
   - For example, in an organization, purchasing department must interact with production department and payroll with personnel department.
3. **Interdependence:**
   - Interdependence means how the components of a system depend on one another.
   - For proper functioning, the components are coordinated and linked together according to specified plan.
   - The output of one sub-system is the required by other sub-system as input.
   - For example, in computer system, the three units Input, System, Output unit are interdependent for proper functioning. No subsystems can function in isolation because it is dependent on data (input) it receives from other subsystems to perform its required tasks.

4. **Integration:**
   - Integration is concerned with how a system is tied together in order to achieve common goal, thus forming integration.
   - It means that the parts of the system work together within the system even if each part performs a unique function.
5. **Central Objective:**
   - The objective of the system must be central.
   - Central objective means the common goal and it may be real or stated. It is not uncommon for an organization to state an objective and operate to achieve another.
   - The users must know the central objective of a computer application in the analysis for a successful design and conversion.
6. **Behavior:**
   - Behavior is the way the system reacts to its surrounding environment.
   - Behavior is determined by the procedures designed to make sure that compon behave in ways that will allow the system to achieve common goal.
7. **Structure:**
   - A relationship among the components which define the boundary between the system and environment is called as the structure of the system.

1.3.3 Basic Components of System

- The functioning units of a system means the basic elements of the system which are interrelated, are the basic components of the system.
- Every system is made up of a set of interrelated elements or basic components. These components are the various parts of a system.
- For example:

Table 1.1

System	Components
Education	Students, Teachers, Library, Laboratory, Buildings, etc.
Computer	Mouse, Display Unit, ALU, Hard Disk, Keyboard, etc.

1.3.4 Elements of System

- All the characteristics of the system are determined by the system elements, their properties and relationships.
- The basic system elements are Input, Processor and Output as shown in Fig. 1.5.
- All systems having common main three components in which all they are described.

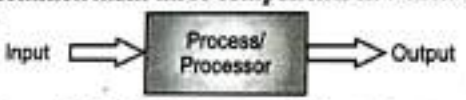


Fig. 1.5: General Model of a System with its Basic Components

1. **Input:**
   - Inputs are elements that make the system to work in order to produce required output.
   - Inputs are the elements that enter the system for processing. The inputs to the system provide all the needed resources to accomplish the goals of the system.
   - Input is defined as energizing or start-up component on which system operates.
   - It may be raw material, data, physical source, knowledge etc., to decide the nature of output.
2. **Processor/Process:**
   - The processes are the action (operation or task) which brings about system goals using the inputs (resources).
   - Processor is the element that involves in the actual transformation of input to output.
   - The processor should be designed of such type that it can accept the input in the given form and can give output in desired format or specification.
   - Sometimes, input is also modified to enable the processor so that it can be handle transformation easily. This modification may be total or partial.
3. **Output:**
   - A major objective of a system is to produce the output as per the user's requirement.
   - Though output largely depends upon the input, its nature, amount, utility, regularity may be different from those of input.
   - Output is defined as, "the result of an input operation after processing".
   - In other words, the outputs of the system include the goals (products or services or new knowledge etc.) which the system is designed.
   - Fig. 1.6 shows an example of the Basic System Model (Payroll System).

Fig. 1.6: Basic Payroll System with its basic components

   - These basic components have certain properties or characteristics. These characteristics affect the operation of the system in speed, accuracy, reliability and capacity.

1.3.5 Information System Environment

- Universal model of a system is made up of system elements like input, processor and output using basic concept like control and feedback to keep the system balance.
- The universal system model is shown in Fig. 1.7.

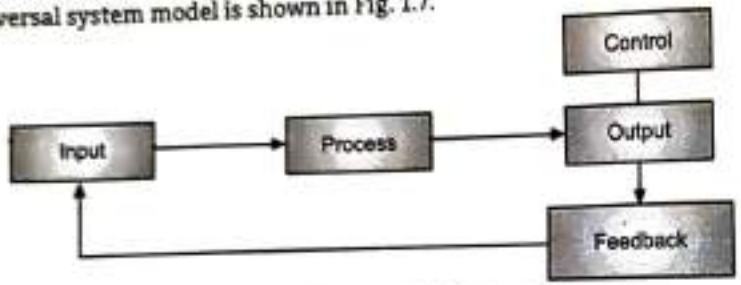


Fig. 1.7: Universal Model of a System

- The major concepts which are essential to build the efficient system & to keep it in balance are: Boundary, environment, Subsystem, Interface, Feedback control, Black box.

1. **Boundary:**
   - The boundary indicates the extent or limit of the system. It divides things into the system and its environment.
   - The elements of the system design its boundary.
   - It is very much essential to limit the system by its boundaries so that system's working can be controlled.
2. **Environment:**
   - All the things which are outside the system are called as environment of the system.
   - The environment affects working or progress of the system.
   - It is a super-system in which any organisation operates.
3. **Subsystem:**
   - Subsystem is a unit that is a part of a larger system. That means the larger system is divided into sub-parts. These sub-parts are again divided into smaller subsystems until the smallest subsystems are of manageable size.
   - For example, computer is system and subsystem contains CPU, Input Unit, Output Unit, and so on.

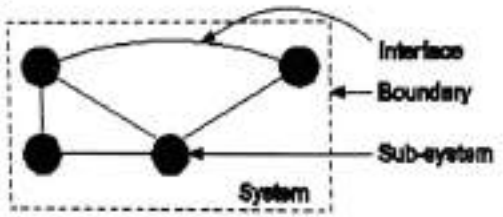


Fig. 1.8: Other Essential Elements of a System

4. **Interface:**
   - Interface means interaction of different system parts with each other as interaction of the system with the system outside its boundaries.
5. **Feedback:**
   - Feedback is a method that helps to compare output produced with output expected and make necessary changes in the process or input in order to reduce the difference between output produced and output expected.
   - It is helpful for decision making in any system. On the basis of feedback, the new business strategies were decided in any organization.
6. **Control:**
   - The control elements guide the system. It controls the working of the system at all stages.
   - It controls all the input, processor and output activities so that decisions can be taken property.
7. **Black Box System:**
   - A subsystem at the lowest level whose processes are not defined is called Black Box system.
   - Only inputs and outputs are known but not actual transformation.

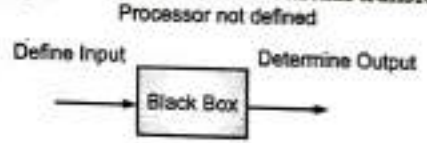


Fig. 1.9: A Black Box System

1.3.6 Types of Systems

- The system is classified in different types on the basis of its analysis. Some types of systems are explained below:

1. **Conceptual (abstract) and Physical Systems**
   - The conceptual system is also known as abstract or analytical system. It is non-physical system.
   - Conceptual (abstract) system is an orderly arrangement of independent ideas.
   - These systems are prepared for studying the physical system. For example: Economic theory, Theory of relativity
   - These systems may be formulas, representation or model of real system.
   - The **Physical Systems** are tangible (countable entities) and could be static or dynamic in nature. Static means which do not change as for working or life of the system is concerned. On the other hand, the term dynamic means, system can be changed due to processing of the system. For example, in computer system the hardware parts are static but the data which changes due to processing is dynamic.
   - **Example:** the computer itself is a physical system and its block diagram is called as abstract system.

2. **Deterministic and Probabilistic Systems**
   - **A deterministic system** works in predictable manner.
   - A Deterministic system is one in which the occurrence of all events is perfectly predictable. If we get the description of the system state at a particular time, the next state can be easily predicted.
   - An example of a deterministic system is the common entrance examination for entry into IIM. All the entities in the system and their interrelationships are well known and given an input the output can be determined with certainty.
   - For example, in Computer system, outputs are deterministic.
   - **A probabilistic System** is one in which the occurrence of events cannot be perfectly predicted.
   - Though the behavior of such a system can be described in terms of probability, a certain degree of error is always attached to the prediction of the behaviour of the system.
   - For example, Weather forecasting system.
3. **Open and Closed Systems**
   - Another classification of system is based on their degree of independence.

   (i) Open System:
   - An open system is a state of a system, in which a system continuously interacts with its environment.
   - It visualizes organizations taking inputs then operations are performed on the input to produce desirable results (output) which are distributed back to the environment.
   - An open system is a one which does not provide for its own control or modification. It does not supervise itself so it needs to be supervised by people.
   - **For example:** If the high speed printer used with computer systems did not have a switch to sense whether paper is in the printer, then a person would have to notice when the paper runs out and signal the system (push a switch) to stop printing.
   - Open systems have input and output flows, representing exchanges of matter, energy or information with their surroundings.

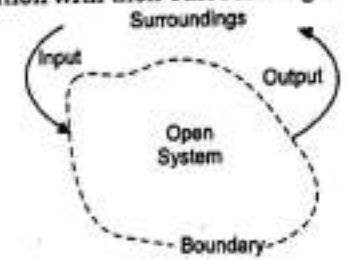


Fig. 1.10: An Open System

- For example, in Education system, student's information is shared with companies for placement.
- Open systems have many interfaces with environment. These system permits interaction across its boundary.

Characteristics: Five important characteristics of open system are:

(a) Input from outside: Open systems are self adjusting and self regulating. When functioning properly, an open system reaches a steady state. Open system can accept input from outside also.

(b) Entropy: Entropy means loss of energy. All dynamic system tends to run down over time resulting in entropy. Open system result entropy by seeking new inputs to return to a steady state.

(c) Process, Output and Cycle: Open system produce useful output and operate in cycles following a continuous flow path.

(d) Differentiation: Open system always increase specialization of functions and greater differentiation of their components. This characteristic offers a completing reason for increasing value of concept of system in system analyst's thinking.

(e) Equifinality: This term implies that the goals are achieved through differing courses of action and a variety of paths. For example, teaching is the goal can be achieved by giving black-board teaching or PowerPoint presentation.

(ii) Closed System:
- A system which does not interact with the outside environment is known as the closed system i.e. it has no input or output.
- A closed system in one which automatically controls or modifies its own operation by responding to data generated by the system itself.
- A closed system is a system in the state of being isolated from the environment.

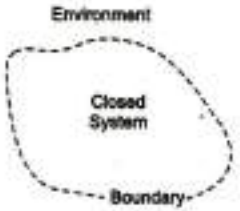


Fig. 1.11: Closed System

- For example, Computer system is a relatively closed system. It does not disturb from outside the system. Chemical reaction in the sealed tube also is an example of closed system.

4. **Natural and Artificial Systems:**
   - Natural systems exist and also bound in nature like rivers, mountains etc., and while artificial systems are manufactured (man-made) such as dams, canals, roads etc.
   - For Example, Solar system.
5. **Man-made Information System:**
   - The systems developed and engineered by humans are a man-made system.
   - Information System (IS) is a man-machine systems relatively closed and deterministic system.
   - Information system may be defined as, "a set of devices, procedures, and operating systems designed around user-based criteria to produce information and communicate it to the user for planning, control and performance."
   - Following pyramid represents different management levels of an organization. This is integrated Man-Machine system **(or Man-made Information System)** that provides information to support planning & in decision making.

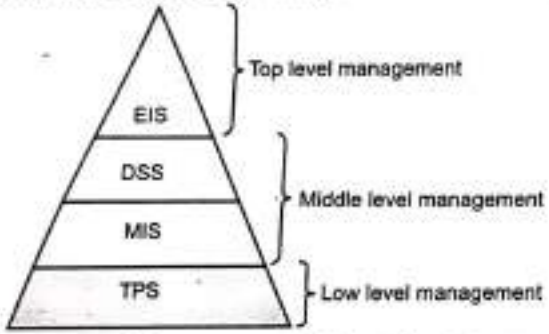


Fig. 1.12: Different types of Information Systems

   - Generally, most of the artificial systems are man-machine systems. For example, a motor bike is a machine system but a bike cannot work without a person.
   - Man-made information systems are divided into three types:

(i) Formal Information System:
- A formal information system is represented by organization chart. The chart is a map of position and their authority relationship indicated by boxes and connected by straight lines.
- It gives a representation of the different parts of system and flow of information in the form of **memos, instructions**, etc., from top level to lower levels of management.

Types of Formal Information System: There are three types of formal information system:

1. **Strategic information system (Top level Management)** relates to long range planning policies. These are the direct interest of the upper management. Policies are the generalization that specify what an organization is going to do. For Example, financial investment in a company, human resources etc.
2. **Management Control system (Middle level Management)** relates to short and intermediate range planning policies. This is done by the direct interest of middle management and department heads for implementation and control.
3. **Operational information system (Bottom level Management)** relates to short term policies. This deals with day-to-day activities like daily employee absence sheets, purchase orders, current stock available for sale.

(ii) Informal Information System:
- This is an employee based system designed to meet personnel and vocational needs and help work related problems.
- Informal information system is related with what is happening practically rather than what is shown on paper.

(iii) Computer Based Information System(CBIS):
- A Computer Based Information System (CBIS) is an information system that uses computer technology to perform some or all of its intended tasks. Today most information systems are computerized, although not all of them are. For this reason the term "Information System" is typically used synonymously with "Computer Based Information System."
- This system is directly dependent on the computer for managing business applications. For example, Automatic Library System, Railway Reservation System, Banking System, etc.
- Computer based information systems are faster, more accurate, more neat and attractive. It is possible to perform different operations easily. Security of data is possible in this system.

Types of CBIS: CBIS can be categorized as follows:

1. **A Decision Support System (DSS):**
   - DSS systems make use of analytical planning modules (operation research model).
   - DSS mostly used for assisting top-level management in decision making. Better decisions are taken by using DSS. DSS reduces clerical work and overtime.
   - DSS also saves cost and time. It consists of decision making with support of other lower level systems (MIS).
   - DSS uses two types of data collected from environment of the organisation:
     - **Internal data:** This type of data used for studying the trends.
     - **External data:** Mostly used for understanding the environment.

Advantages of DSS:

1. It helps to improve personal efficiency and problem solving.
2. It helps to increase organisational control.

Examples:

- Bank loan management systems.
- Financial planning systems.

2. Transaction Processing System (TPS):

- This is the most fundamental computer based system in an organization. This relates to the processing of business transactions.
- A transaction processing system can be defined as a system that captures, classifies stores, maintains, updates and retrieves transaction data for record keeping and input to the other types of CBIS.
- Transaction Processing System is targeted at improving the routine business activities. A transaction is any event or activity that affects the whole organization.

Examples:

- **Airline booking systems:** Flights booking management.
- **Point of Sale Systems:** Records daily sales.
- **Payroll systems:** Processing employee's salary, loan's management, etc.
- **Stock Control systems:** Keeping track of inventory levels.

3. Management Information System (MIS)

- A MIS is a system that provides historical information, information on the current status. It is a communication process in which data are recorded and processed for further operational uses.
- An MIS is a set of Computer Based System and procedures implemented to help managers in their essential job of decision making. The actual process will involve the collection, organization, distribution and storage of organization wide information for managerial analysis and control.
- MIS is made up of three components:
  - (a) **Management:** Emphasizing the ultimate use of such information system for decision making.
  - (b) **Information:** Information highlighting on processed data rather than the raw data and in the context in which managers and other end users use it.
  - (c) **System:** Highlighting a fair degree of integration.
- The MIS system investigates the input with routine algorithms i.e. aggregate, compare and summarizes the results to produced reports that tactical managers use to monitor, control and predict future performance.

- For example, input from a Point-of-Sale system can be used to analyze trends of products that are performing well and those that are not performing well. This information can be used to make future inventory orders i.e. increasing orders for well-performing products and reduce the orders of products that are not performing well.

Examples:

- **Human Resource Management System:** Overall welfare of the employees, staff turnover, etc.
- **Sales Management Systems:** These systems get input from the point of sale system.
- **Budgeting Systems:** These systems give an overview of how much money is spent within the organization for the short and long terms.

(i) Executive Support System (ESS) or Executive Information System (EIS)

- This system also called an Executive Information System (EIS). It is made especially for top managers that specifically supports strategic decision making.
- EIS is structured tracking system. It provides rapid access to timely information and direct access to management reports.
- EIS contains extensive graphics capabilities. It serves the information needs of top executives. It gives quick and easy access to detailed information.

Advantages of EIS:

1. EIS is easy for upper level executives.
2. It provides timely delivery of company summary information.
3. It improves to tracking information.
4. It filters data for management.
5. It offers efficiency to decision making.

1.4 PROJECT ORGANIZATION

- Project organization is a process, which provides the arrangement for decisions on how to realize a project.
- It decides the project's process: planning how its costs, deadlines, personnel, and tools will be implemented.
- The project organization is then presented to the project stakeholders.
- The project organization is the structure of step by step planning of the project to build.
- It plays an important role in success of project to be implemented.
- It is created separately, with specialists and workers from various departments in the organization.
- These personnel work under the project manager.

1.4.1 Project Organizational Structure Charts

- Project organization structure chart is path to the start organizing a project.
- Project organization structure chart is real work of implementation and application to project organization. So a project organization chart is so important.
- It establishes the formal relationships between the project manager, project team, development organization, the project itself and project stakeholders.
- The project organization chart will identify the roles and responsibilities of the team.
- This includes identifying training if needed, recognizing how to allocate resources and determining appropriate ways to involve stakeholders.

How to organize a project?

- To perform above mentioned activities, there are six steps to take as follows:

Step 1: Identify Personnel

- First of all, identify the people those who have an impact on the project. The one who are related to the project scope.
- They are the key staff.
- These people are from different areas like marketers to salespeople, department heads and IT personnel to consultants and support staff, etc.

Step 2: Create Senior Management Team

- The next step is to get a team who is responsible for the project.
- These are individuals with an interest in the project and are committed to its success.
- This team is usually made up of **project sponsors** or the client, though it can also include experts who offer guidance throughout the project.

Step 3: Assign Project Coordinators

- There is a need to have a communication person, or group at the mid-to-low management level, to carry out duties that fall to this level.
- This person or group will help synchronize team tasks.
- The number of coordinators will be determined by the size of the project, but always focus on three areas of a project: Planning, Technical and Communications.

Step 4: Identify Stakeholders

- Apart from team that will execute the project, it is a key to identify the stakeholders, as they are also impacted by the project and participate in the project development.

Step 5: Establish Training Requirements

- Sometimes teams are efficient in their tasks and with the tools that have been furnished to help them.
- So they are in need a period of training before the project can be executed.

- This is the point where any training that is needed is established and offered to the team.
- The project coordinator is usually the person who manages this task of training team members.

Step 6: Develop the Project Organization Chart

- The final step is to develop the project organization chart.
- First, review the previous steps and then make this visual representation of how the people in the project will collaborate, what their duties are and where they're interrelated.
- To develop a project organization chart, you may use a free network diagram tool, such as Google Draw, and when done then distributed it to the necessary parties.

- The project organization chart must have identified the primary decision-makers and their authorities.
  - Each person involved in the project must have an assignment and identified role and the responsibilities of those roles clearly defined. Any links connecting roles must be identified, as well as all the stakeholders.
  - Be sure that the reporting and communications channels are also defined and described.

1.5 COMMUNICATION IN PROJECT MANAGEMENT

- Project communication management is a collection of processes that help make sure the right messages are sent, received, and understood by the right people.
- Project communication management is one of the 10 key knowledge areas in the PMBOK (Project Management Book of Knowledge). The processes included in this area have changed over the years but, in the current version, there are three primary project communication management processes.
- These are:
  1. Plan communications management
  2. Manage communications
  3. Monitor communications

1.5.1 How to Create a Project Communication Management Plan?

- A project communication management plan documents how the project manager manages and controls communication across their projects.
- When creating a plan, project managers should follow these five steps:
  1. **Decide project communication plan objectives:** What will be the purpose of your communication? You may use some communication tools for awareness, such as a status report. Others may require action, such as requiring a sponsor to authorize spending or a customer to approve project testing.

2. **Determine your audience:** Who are the **stakeholders** in this project? You should make an extensive list of everyone involved. Consider anyone impacted by the project or who influences its success. This list should include team members, sponsors, customers, and other interested parties.
3. **Write your message:** Identify communication necessary to satisfy stakeholder expectations and keep them informed. The message for each type of communication is the actual content that will be shared. Key components to be communicated include scope, **schedule**, budget, objectives, risks, and **deliverables**.
4. **Choose your channel:** Identify how the message will be delivered. Will it be a formal report emailed out to all stakeholders? Or will it be an informal verbal question during a team meeting?
5. **Set a timeline:** Identify time-frame and/or frequency of communication messages. When will you deliver your message? Do your stakeholders require weekly or monthly reports? Is there a deadline to meet? Consider varying time zones and employee schedules here.

- Project communication management plan should be detailed enough to lay out why you are sending a message, to whom you are sending it, what specific information will be sent, how you are going to send it, and when.
- Involving your stakeholders in the creation of this plan is important. You need to understand their communication preferences and expectations. If you over-communicate, they may stop paying attention. But, if you under-communicate, it can lead to misunderstandings and issues.
- The golden rule is that, to be a good communicator and a good listener. Pay attention to all the factors and take every opinion into account before creating your project communication management plan.

1.5.2 Manage Project Communication

- Once the project communication management plan has been created and approved, it's the project manager's job to ensure it's carried out successfully. This means the plan needs to be reviewed and updated on a regular basis to reflect any changes to the project or its stakeholders.
- The project manager also has to manage the execution of the project communication management plan. This includes:
  1. Collection and analysis of data.
  2. Creation of messages for communication.
  3. Transmission or distribution of communications.
  4. Storage of any communication reports, files, or documents.
  5. Retrieval of any stored communications.
  6. Disposal of any old communications upon project closure or a set date.

1.5.3 Monitor Project Communication

- This process is called 'control communications. 'Control Communications' process has been renamed as 'Monitor Communications' in PMBOK (Project Management Body of Knowledge) Guide - 6th version.
- It is the process of monitoring and controlling project communications throughout the entire project lifecycle to ensure the information needs of project stakeholders are met.
- This may include the confirmation of the following:
  - Communications went out as planned.
  - They were received by the proper stakeholders.
  - Messages were understood.
  - Any relevant feedback was provided to the appropriate project members.
- The actual type of monitoring, including method and frequency, should be a part of the project communication management plan.

1.6 RISK MANAGEMENT IN PROJECT MANAGEMENT

Introduction:

- Project risk management is the process that project managers use to manage potential risks that may affect a project in any way, both positively and negatively. The goal is to minimize the impact of these risks.
- A risk is any unexpected event that can affect people, technology, resources, or processes (including projects).
- Like a regular problem that may arise, risks may occur suddenly, sometimes unexpectedly.
- Risk management is applicable to large and small projects in different ways.
- In large-scale projects, detailed planning for each risk may be done if issues arise.
- For smaller projects, risk management might mean a simple, prioritized list of high, medium and low priority risks.
- Risk can be either positive or negative, though most people assume risks are inherently the latter.
- Negative risk is the one which may damage a project.
- Positive risks are opportunities that can affect the project in beneficial ways.
- Project management software can help you keep track of risk. Assign team members to own those risks, add documents, set priority and more.

1.6.1 How to manage Risk?

- In risk management, the objectives to be delivered for the project should be precisely defined.
- In other words, write a very detailed project charter, with your project vision, objectives, scope and deliverables.

- This way risks can be identified at every stage of the project. Then you will want to engage your team early in identifying any and all risks.
- Many project managers simply email their project team and ask to send them things they think might go wrong on the project.
- But to better identify project risk, all stakeholders must be involved, that is entire project team, your clients' representatives, and vendors into a room together and do a risk identification session.
- With every risk you define, you'll want to log it somewhere—using a risk tracking template helps you prioritize the level of risk.
- Next step is create a risk management plan to capture the negative and positive impacts on the project and what actions needed to take to deal with them.

1.6.2 Risk Management Process

- Following are six steps in the Risk Management Process as shown in Fig. 1.13.

Fig. 1.13: Risk Management Framework

Step 1: Identify the Risk

- To identify risk, there are many ways. By following this step, we may collect the data in a risk register.
- One way is discussion with project team, colleagues and stakeholders.
- Find the experts with relevant experience and set up interviews so you can gather the information you'll need to both identify and resolve the risks.
- Identify negative risks. Note them. Study existing projects to know their negative risks. Now your list of potential risks has grown.
- Confirm that the identified negative risks are the rooted in the cause of a problem.

Step 2: Analyze the Risk

- Analyzing risk is very difficult. There is never enough information you can gather.
- Of course, a lot of that data is complex, but most industries have best practices, which can help you with your analysis.

- After studying project risk, it may list impacts, such as avoiding potential litigation, addressing regulatory issues, complying with new legislation, reducing your exposure and minimizing impact.
- Through qualitative and quantitative risk analysis, you can determine how the risk is going to impact your schedule and budget.
- Project management software helps you analyze risk by monitoring your project. .

Step 3: Prioritize the Risk

- Evaluate the risk to know which resources need to assemble towards resolving it when and if it occurs.
- Immediate attention is required to certain risks. These are the risks that can derail your project.
- Failure isn't an option. Low priority risks are important. Then there are those risks that have little to no impact on the overall project's schedule and budget.

Step 4: Assign an Owner to the Risk

- Identify the person who is responsible for handling that risk.
- There might be a team member who is more skilled or experienced in the risk. Then that person should lead the charge to resolve it.
- Assign the task to the right person, making sure that every risk has a person responsible for it.

Step 5: Respond to the Risk

- Once you found a risk, then you need to know if this is a positive or negative risk.
- For each major risk identified, create a plan to follow it. Develop a strategy which is a preventative or contingency plan.
- Find solution for the risk which is prioritized. Communicate with the risk owner and together decide on which of the plans you created to implement to resolve the risk.

Step 6: Monitor the Risk

- Track the progress towards resolution of the identified risk whoever owns the risk.
- Risk owner need to stay updated to have an accurate picture of the project's overall progress to identify and monitor new risks.
- You will want to set up a series of meetings to manage the risks. Make sure you have already decided on the means of communications to do this.
- Proper communication channels are play vital role .It is best if everyone in the project knows what is going on, so they know what to be on the lookout for and help manage the process.

1.6.3 Gantt Charts for Risk Management Plans

Fig. 1.14: Gantt Chart

- Above Fig. 1.14 represents Gantt chart. The Gantt chart is mainly a kind of horizontal bar chart that measures the job completion time versus the time designated for that job's completion.
- Gantt charts also include components that show which resources are utilized and how the assignments line up with each other. The chart offers a graphic illustration that makes it easy to coordinate specified tasks.
- Gantt charts are used to create detailed risk management plans to prevent risks from becoming issues.
- Schedule, assign and monitor project tasks with full visibility. Team members can even add comments and files to their assigned tasks, so all the communication happens on the project level.

Summary

- A system development methodology is an orderly and integrated collection of various methods, tools & techniques.
- A software life cycle model is a particular abstraction that represents a software life cycle. A software life cycle model is often called a Software Development Life Cycle (SDLC).
- SDLC concentrates on feasibility analysis, cost benefit analysis, project management hardware and software selection and personnel consideration.

- There are two phases within SDLC:
  1. System Analysis
  2. System Design
- There are three types of prescriptive process models:
  1. The waterfall model often used with well-defined adaptations or enhancements to current software.
  2. The incremental model is defined as, "a model of software development where the product is designed, implemented and tested incrementally (a little more is added each time) until the product is finished."
  3. Evolutionary model is useful for projects using new technology that is not well understood. This is also used for complex projects where all functionality must be delivered at one time, but the requirements are unstable or not well understood at the beginning. It has basic two models i.e. Prototyping and Spiral Model.
- A system is defined as, "a collection of related components that interact to perform a task in order to accomplish a goal".
- The basic system elements are Input, Processor and Output.
- The system is classified in different types on the basis of its analysis. Some types of systems are:
  1. Conceptual (abstract) and Physical Systems
  2. Deterministic and Probabilistic Systems
  3. Open and Closed Systems
  4. Natural and Artificial Systems
- A Computer Based Information System (CBIS) is an information system that uses computer technology to perform some or all of its intended tasks.
- Project organization is a process which provides the arrangement for decisions on how to realize a project.
- Project organization structure chart is real work of implementation and application to project organization. So a project organization chart is so important.
- Project communication management is a collection of processes that help make sure the right messages are sent, received, and understood by the right people.
- Project risk management is the process of identifying, analyzing and responding to any risk that arises over the life cycle of a project to help the project remain on track and meet its goal.

Check Your Understanding

1. A system _____ methodology is an orderly and integrated collection of various methods, tools and techniques.
   (a) development (b) software
   (c) hardware (d) middleware
2. Long form of SDLC is _____.
   (a) Software Development Life Cycle (b) System Development Life Cycle
   (c) Software Do Life Cycle (d) Software Development Cycle
3. There are two phases within SDLC i.e. System Analysis and _____.
   (a) System development (b) System Design
   (c) System testing (d) System maintenance
4. Evolutionary model has basic two models i.e. Prototyping and _____.
   (a) Spiral Model (b) Water fall mode
   (c) Incremental model (d) Agile model
5. A _____ is defined as, "a collection of related components that interact to perform a task in order to accomplish a goal".
   (a) System (b) Software
   (c) Hardware (d) Machine
6. The basic system elements are Input, _____ and Output.
   (a) Control (b) Processor
   (c) Device (d) ALU
7. _____ is an information system that uses computer technology to perform some or all of its intended tasks.
   (a) A Computer Basic Information System (CBIS)
   (b) A Computer Basis Information System (CBIS)
   (c) A Computer Based Information Software (CBIS)
   (d) A Computer Based Information System (CBIS)
8. Project _____ is a process, which provides the arrangement for decisions on how to realize a project.
   (a) organization (b) management
   (c) communication (d) risk management
9. Project organization structure chart is real work of _____ and application to project organization.
   (a) interaction (b) presentation
   (c) implementation (d) communication

10. Project communication management is a collection of _____ that help make sure the right messages are sent, received, and understood by the right people.
    (a) information (b) data
    (c) process (d) processed data
11. Project risk management is the process of identifying, analyzing and responding to any _____ that arises over the life cycle of a project to help the project remain on track and meet its goal.
    (a) risk (b) communication
    (c) organization (d) none of these

Answers

1. (a)	2. (b)	3. (b)	4. (a)	5. (a)	6. (a)	7. (b)	8. (d)	9. (a)	10. (c)	11. (a)

Practice Questions

Q.I Answer the following questions in short.

1. What is SDLC?
2. Which are system characteristics?
3. What is the Spiral Model?
4. Which are phases of waterfall model?
5. Which are the types of system process model?
6. What is open and closed system?
7. Write a short note on Evolutionary model.

Q.II Answer the following questions.

1. Define SDLC and explain phases in SDLC.
2. What is System analysis? Explain Types of SDLC Process Model.
3. Draw and explain Prototyping Model.
4. Explain Spiral model with well labelled diagram. Give its advantages and disadvantages.
5. Explain Waterfall model with well labelled diagram. Give its advantages and disadvantages.
6. Explain Incremental Process model with well labelled diagram. Give its advantages and disadvantages.
7. Explain Prototyping model with well labelled diagram. Give its advantages and disadvantages.
8. Draw and explain components of system.
9. What is project organization? Explain steps to perform activities in project organization.
10. Explain Communication in Project Management.
11. What is Risk management in Project Management?

Q.III Define the terms.

1. SDLC
2. System
3. Risk Management
4. Project Organization
5. MIS

2... SRS Documentation

Learning Objectives ...

- To know about software requirements specification.
- To learn about Customer Requirement Specifications (Elicitation).

2.1 INTRODUCTION

- A Software Requirements Specification (SRS) is a document that captures complete description about how the system is expected to perform. The SRS needs to be correct, complete, consistent, updatable, verifiable etc. It is usually signed off at the end of requirements engineering phase.
- SRS contains a contract between the customer and the developer. This SRS document is used for verifying whether all the functional and non-functional requirements specified in the SRS are implemented in the product.
- Software Requirement specification activity has three inter-related parts.
  1. **Analysis of Factual Data:** The data collected during the fact finding study and included in data flow and decision analysis documentation are examined to determine how well the system is performing and whether it will meet the organization's demands.
  2. **Identification of Essential Requirement:** Features that must be included in a new system, ranging from operational details to performance criteria are specified.
  3. **Selection of Requirement Strategies:** The methods that will be used to achieve the stated requirement are selected. These form the basis for system design, which follows the requirement specification.
- All these three activities are important and must be performed correctly.

2.1.1 Characteristics of SRS

Various characteristics of SRS are:

- **Correct:** The SRS should be made-up to date when appropriate requirements are identified.
- **Unambiguous:** When the requirements are correctly understood then only it is possible to write an unambiguous SRS.

- **Complete:** To make the SRS complete, it should be specified what a software designer wants to create a software.
- **Consistent:** It should be consistent with reference to the functionalities identified.
- **Specific:** The requirements should be mentioned specifically.
- **Traceable:** What is the need for mentioned requirement? This should be correctly identified.

2.1.2 IEEE Standard Format of SRS

- The IEEE Std.830-1998 was created to standardize the software requirements specification document.
- The aim of an SRS document is to capture requirements in an unambiguous manner in order to facilitate communication between stakeholders.
- IEEE Std.830-1998 provides a structure (template) for documenting the software requirements.
- The SRS document described in IEEE Std.830 is divided into a number of recommended sections to ensure that information relevant to stakeholders is captured.
- This specification document serves as a reference point during the development process and captures requirements that need to be met by the software product.
- Basic issues addressed in the SRS include functionality, external interfaces, performance requirements, attributes and design constraints.
- SRS also serves as a contract between the supplier and customer with respect to what the final product would provide and help achieve.
- Although the IEEE Std. 830-1998 specifies the structure it does not choose one representation for requirements over the other. Neither does it specify what techniques should be used to populate the various sections.

2.1.3 SRS Format

- SRS is the standard statement of what the system developers should implement.
- SRS includes the user's requirements for a system and a detailed specification of the system requirement.
- Fig. 2.1 shows the structure of SRS document.

Section 1	:	Product Overview and Summary
Section 2	:	Development, Operating, and Maintenance Environments
Section 3	:	External Interfaces and Data Flow
Section 4	:	Functional Requirements
Section 5	:	Performance Requirements
Section 6	:	Exception Handling
Section 7	:	Early Subsets and Implementation Priorities
Section 8	:	Foreseeable Modifications and Enhancement
Section 9	:	Acceptance Criteria
Section 10	:	Design Hints and Guidelines
Section 11	:	Cross-reference Index
Section 12	:	Glossary of Terms

Fig. 2.1: Structure of SRS Document

- SRS format's sections are described below:
  - **Section 1 and 2** of the SRS document present an overview of product features and summarize the processing environments for development, operation, and maintenance of the software product.
  - **Section 3** of the SRS document specifies the externally observable characteristics of the software product and it includes user displays and report formats, a summary of user commands and report options, data flow diagrams, and a data dictionary.
  - **Section 4** of the SRS document specifies the functional requirements for the software product. Typically, functional requirements are expressed in relational and state-oriented notations that specify relationships among inputs, actions and outputs etc.
  - **Section 5** of the SRS document specifies the performance characteristics like response time for various activities, processing time for various processes, throughput, primary and secondary memory constraints, required telecommunication bandwidth, and unusual reliability requirements etc.
  - **Section 6** of the SRS document specifies the exception handling, including the actions to be taken and the messages to be displayed in response to undesired situations or events or errors. Various categories of possible exceptions include temporary and permanent resource failure, incorrect, inconsistent or out range input data, violation of capacity limits and violations of restrictio operators etc.
  - **Section 7** of the SRS document specifies early subsets and implemen priorities for the system under development and it is important to sp implementation priorities for various system capabilities.
  - **Section 8** of the SRS document specifies foreseeable modifications and enhancements that may be incorporated into the product following initial product release.
  - **Section 9** of the SRS document specifies the software product acceptance criteria. Acceptance criteria specify functional and performance tests that must be performed, and the standards to be applied to source code, internal documentation and external documentations like the design specifications, the test plan, the user's manual, the principles of operations, and the installation and maintenance procedures and so on.
  - **Section 10** of the SRS document specifies design hints and guidelines. The "how to" of product implementation is the topic of software design, and should be deferred to the design phase of product.
  - **Section 11** of the SRS document specifies product requirements to the source of information used in deriving the requirements.
  - **Section 12** of the SRS document specifies the definition of terms that may be unfamiliar to the customer and the product developers. Proper care should be taken to define standard terms that are used in non-standard ways.

2.2 SRS SPECIFICATION

Types of Specification:

- The types of requirements which are captured during SRS are as follows:

Fig. 2.2: Types of Specification

- Mainly, a software requirement can be of 3 types:
  1. **Functional requirements:** Functional requirements are statements or goals used to define system behavior. Functional requirements define *what* a software system must do or not do. They are typically expressed as responses to inputs or conditions.
  2. **Non-functional requirements:** Non-functional requirements relate to software usability. Non-functional software requirements define *how* the system must operate or perform. A system can meet its functional requirements and fail to meet its non-functional requirements.They cover performance, usability, scalability, security, portability.
     They basically deal with issues like:
     - Portability
     - Security
     - Maintainability
     - Reliability
     - Scalability
     - Performance
     - Reusability
     - Flexibility

  3. **Domain requirements:** Domain requirements are the requirements which are characteristic of a particular category or domain of projects. Domain requirements can be functional or nonfunctional.

2.3 REQUIREMENT ELICITATION

- Elicitation means asking the customer, the user, and other people about the objectives for system or product.
- Eliciting requirements is difficult because of:
  1. **Problems of scope** in identifying the boundaries of the system or specifying unnecessary technical detail rather than overall system objectives.
  2. **In Problems of understanding** what is needed, what the problem domain is, and what the computing environment can handle (Information that is believed to be "obvious" is often omitted).
  3. **Problems of Volatility** due to the requirements change over time.
- Requirements gathering (also called requirements elicitation) combines all the aspects of problem solving, elaboration, negotiation, and specification.
- In order to increase the collaborative, team-oriented approach to requirements gathering, all the stakeholders work together to identify the problem, propose elements of the solution, negotiate and compromise on different approaches and then specify a preliminary set of solution requirements.

(a) Collaborative Requirements Gathering:

- Today, many different approaches to collaborative requirements gathering have been proposed. Each and every use of a slightly different scenario, but all apply some variation on the following basic guidelines:
  (i) Rules for preparation and participation are established.
  (ii) Meetings are conducted and attended by both software engineers and other stakeholders.
  (iii) A "facilitator" (can be a customer, a developer etc.) controls the meeting.
  (iv) An agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas.
  (v) A "definition mechanism" (can be work sheets, flip charts, chat room, etc.) is used.
- The goal is to identify the problem, propose elements of the solution, negotiate different approaches, and specify a preliminary set of solution requirements in an atmosphere i.e., conducive to the accomplishment of the goal.
- To better understand the flow of events as they occur, we present a brief scenario that outlines the sequence of events that lead up to the requirements gathering meeting, occur during the meeting, and follow the meeting.
- During inception basic questions and answers establish the scope of the problem and the overall perception of a solution.

(b) Quality Function Deployment (QFD):

- QFD is a quality management technique that translates the needs of the customer into technical requirements for software.

- QFD concentrates on maximizing customer satisfaction from the software engineering process.
- To accomplish this, QFD emphasizes an understanding of what is valuable to the customer and then deploys these values throughout the engineering process.
- QFD identifies three types of requirements as explained below:

(i) **In Normal Requirements,** the objectives and goals that are stated for a product or system during meetings with the customer. If these requirements are present, the customer is satisfied.
Examples of normal requirements might be requested types of graphical displays, specific system functions, and so on.

(ii) **Expected Requirements** are implicit to the product or system and may be so fundamental that the customer does not explicitly state them. Their absence will be a cause for significant dissatisfaction.
Examples of expected requirements are ease of human/machine interaction, overall operational correctness and reliability and so on.

(iii) **Exciting Requirements** features go beyond the customer's expectations and prove to be very satisfying when present.
For example, software for a new smart phone comes with standard features, but is coupled with a set of unexpected capabilities (like multi-touch screen) that delight every user of the product.

- Although, QFD (Quality function deployment) concepts can be applied across the entire software process. However, specific QFD techniques are applicable to the requirements elicitation activity.
- QFD uses customer interviews and observation, surveys, and examination of historical data (e.g., problem reports) as raw data for the requirements gathering activity. These data are then translated into a table of requirements called the customer voice table i.e., reviewed with the customer and other stakeholders. A variety of diagrams, matrices, and evaluation methods are then used to extract expected requirements and to attempt to derive exciting requirements.

(c) **Usage Scenarios:**

- As requirements in requirement gathering are gathered, an overall vision of system functions and features begins to materialize. However, it is difficult to move into more technical software engineering activities until you understand how these functions and features will be used by different classes of end users.
- To accomplish this, developers/engineer and users can create a set of scenarios that identify a thread of usage for the system to be constructed. The scenarios, often called use cases, provide a description of how the system will be used.

(d) **Elicitation Work Products:**

- The work products produced as a consequence of requirements elicitation will vary depending on the size of the system or product to be built.

- For most systems, the work products include following:
  (i) A bounded statement of scope for the system or product.
  (ii) A statement of need and feasibility.
  (iii) Any prototypes developed to better define requirements.
  (iv) A set of usage scenarios that provide insight into the use of the system or product under different operating conditions.
  (v) A list of customers, users, and other stakeholders who participated in requirements elicitation.
  (vi) A description of the system's technical environment.
  (vii) A list of requirements (preferably organized by function) and the domain constraints that applies to each.

2.4 BUSINESS ENGINEERING

- **Business Engineering (BE)** is the development and implementation of business solutions, from business model to business processes and organizational structure to information systems and information technology (From Wikipedia).
- Business engineering focuses on developing innovative business solutions that take a Socio-Technical Systems (STS) approach. The STS approach addresses an enterprise as a total system in the same manner as an airplane system or an industrial facility, since they possess a similar level of complexity.
- Business engineering combines knowledge in the fields of business administration, Industrial Engineering, as well as information technology and connects it to all aspects of transformation, from means of presentation to process models to cultural and political considerations.
- Business engineering is a technique in which traditional engineering principles are applied to the business world. This technique relies heavily on science and maths, which is in direct contrast to traditional business models, many of which are largely theoretical.
- The goal of business engineering is to produce measurable results or quantifiable data rather than just an arbitrary improvement or change. Unlike many other business models, business engineering focuses on a holistic approach to problem solving. Rather than simply address different aspects of the business individually, business engineering professionals concentrate on the interaction between different factors within the company and how they impact one another.

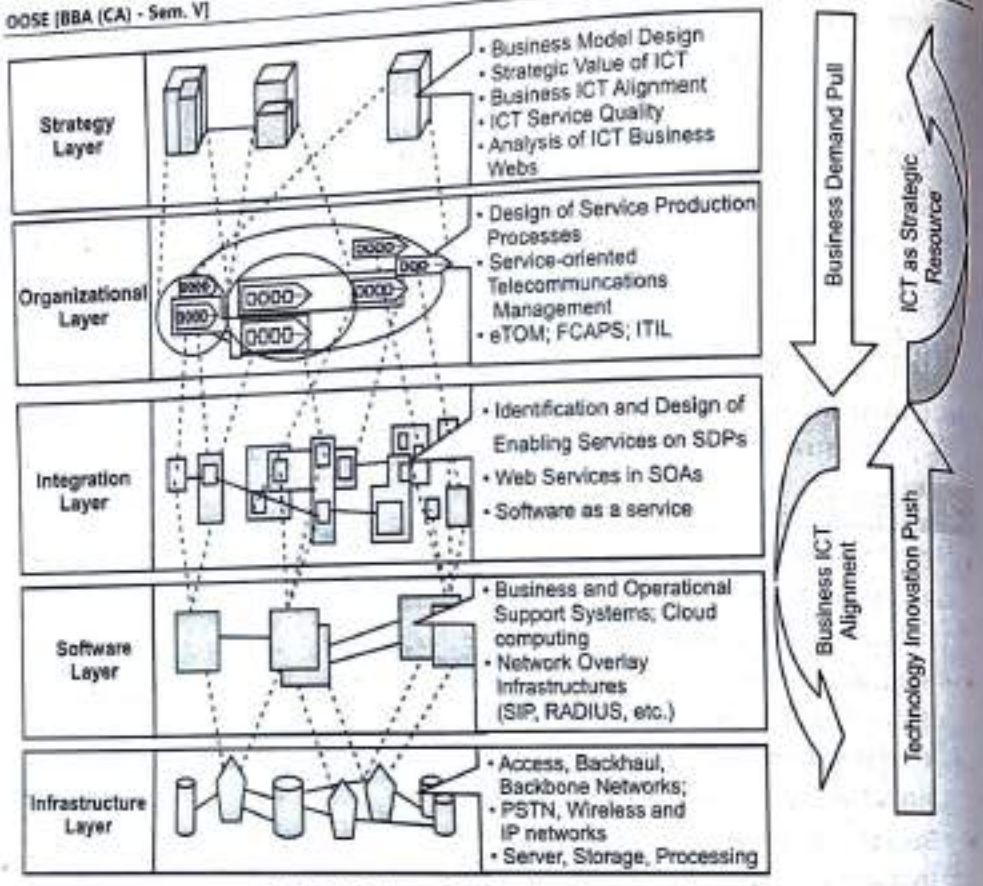


Fig. 2.3: Business Engineering Framework

- The business engineering framework distinguishes artefacts and models on a Stategy, Organisational, Integration, Software/Data and IT Infrastructure layers (Fig. 2.3).
  - The **Strategy** layer describes the positioning of an enterprise at a high level of abstraction. It is developed once the business strategy is defined.
  - The **Organizational** layer specifies a firm's organizational structure and its process organization. It includes static (structural) as well as dynamic (flow) aspects. This framework focuses on IS as an enabler of organizational change and business process redesign.
  - The **Integration** layer describes how applications share or could share data and functions with other applications and databases.
  - The **Software/Data layer** determines the data elements and software applications that support the business layer.
  - The IT **Infrastructure** layer comprises the hardware platforms and communication infrastructure that supports the applications.

Summary

- A Software Requirements Specification (SRS) is a document explaining how and what the software/system will do.
- It is organized into independent sections and each section is organized into modules or units.
- Elicitation means asking the customer, the user, and other people about the objectives for system or product.
- QFD is a quality management technique that translates the needs of the customer into technical requirements for software.
- Business Engineering (BE) is the development and implementation of business solutions, from business model to business processes and organizational structure to information systems and information technology.

Check Your Understanding

1. A software requirements specification (SRS) is a _____ that captures complete description about how the system is expected to perform.
   (a) document (b) software
   (c) system (d) specification
2. SRS contains a _____ between the customer and the developer.
   (a) contract (b) understanding
   (c) meeting (d) specification
3. If every requirement stated in the Software Requirement Specification (SRS) has only one interpretation, SRS is said to be _____.
   (a) Correct (b) Unambiguous
   (c) Verifiable (d) Consistent
4. _____ means asking the customer, the user, and other people about the objectives for system or product.
   (a) SRS (b) Elicitation
   (c) Specification (d) None of the mentioned
5. The major goal of requirement determination phase of information system development is _____.
   (a) Determine whether information is needed by an organization.
   (b) Determine what information is needed by an organization
   (c) Determine when information is to be given.
   (d) Determine how information needed by an organization can be provided.
6. Which one of the following is not a step of requirement engineering?
   (a) Requirement analysis (b) Requirement elicitation
   (c) Requirement design (d) Requirement documentation

7. Which of the following objectives is not the primary objective in the preliminary investigation of system development?
   (a) Assess cost and benefit of alternative approves.
   (b) Determining the size of the project.
   (c) Preparing the SRS to cover all the system specifications.
   (d) Report finding to the management with recommendation to accept or reject the proposal.
8. Business engineering (BE): is the development and implementation of _____.
   (a) business solutions (b) software engineering
   (c) software (d) requirement specification
9. Business engineering combines knowledge in the fields of business administration, Industrial Engineering, as well as _____ and connects it to all aspects of transformation.
   (a) technology (b) information technology
   (c) information (d) software engineering
10. The work products produced as a consequence of requirements elicitation will vary depending on the _____ of the system or product to be built.
   (a) size (b) length
   (c) height (d) width

Answers

1. (a)	2. (a)	3. (b)	4. (b)	5. (b)	6. (c)	7. (c)	8. (a)	9. (b)	10. (a)

Practice Questions

Q.I Answer the following questions in short.

1. What is SRS?
2. What is requirement elicitation?
3. What is business engineering?
4. What is QFD?
5. What is the use of Section 4 in SRS format?

Q.II Answer the following questions.

1. Explain IEEE Standard format of SRS in detail.
2. Explain what is requirement elicitation? Explain in detail.
3. Explain what is business engineering? Explain it with the help of business engineering framework diagram.
4. Describe the concept of collaborative requirements gathering in detail.
5. Give reasons why eliciting requirement is difficult?

Q.III Define the terms.

1. SRS
2. Requirement elicitation
3. Business engineering

■■■

3... Introduction to UML

Learning Objectives ...

- To know about Unified Modeling Language (UML).
- To study reasons for learning UML.
- To learn about three building blocks of UML.
- To get information on different types of UML diagrams.
- To know different Things and Relationships in UML.
- To study UML rules and UML Architecture.
- To get knowledge of UML Advantages and disadvantages.

3.1 CONCEPT OF UML [S-18, W-18, S-19]

- UML stands for **Unified Modeling Language**. UML is a standard graphical language for modeling object-oriented software.
- UML can be described as a general-purpose visual modeling language which is used to visualize, specify, construct, and document software systems.
- UML is not a programming language, but tools can be used to generate code in various languages using UML diagrams.
- UML was created by Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997. It was developed in the mid-1990s as a collaborative effort by James Rumbaugh, Grady Booch and Ivar Jacobson.

Definition:

- It can be defined as "UML is a language for visualizing, specifying, constructing and documenting artifacts of a software intensive system".
- In this definition,
  - **Visualizing** means UML has the standard diagramming notations for drawing or presenting pictures of software systems.
  - **Specifying** means building models that are exact, unambiguous and complete.
  - **Constructing** is related to actually implementation of design into coding.
  - **Documentation** plays a vital role in any type of system development, which helps during developing a system and after its deployment. UML addresses documentation of system architecture and other details.

3.1.1 Goals of UML

Following are main goals of UML:

- Provide users with a ready-to-use, expensive visual modeling language so they can develop and exchange meaningful models.
- Provide extensibility and specialization mechanisms to extend the central concepts.
- Be independent of particular programming languages and development processes.
- Provide a formal foundation for understanding the modeling language.
- Encourage the growth of the Object Orientation tools market.
- Support higher-level development concepts such as collaborations, frameworks, patterns and components.
- Integrate best practices.

3.2 REASONS FOR THE USE OF UML

- There are various reasons for the use of UML. Some of them are listed below:
  1. **Software construction needs a plan:** Software development is extremely complex and critical. Once a particular structure is in place, it is often a lengthy and difficult job to change. UML helps in constructing a model.
  2. **Appropriate for both new and legacy systems:** UML is appropriate for both new system developments and improvements to existing systems. Only the parts that are modeled will be affected by the change.
  3. **Inherent traceability:** The Use Case Driven nature of modeling with UML ensures that all levels of model trace back to elements of the original functional requirements.
  4. **Parallel development of large systems:** Large complex, critical UML models can be decomposed in a totally user-definable way so that, different parts of the model can be developed independently by different people or groups.
  5. **Visualize in multiple dimensions and levels of detail:** UML allows software to be visualized in multiple dimensions, so that a computer system can be completely understood before construction/development begin.
  6. **Incremental development and redevelopment:** UML models respond well to an incremental development environment.
  7. **Unified:** OMG (Object Management Group) an independent organization has made UML as the de-facto (actual) standard modeling language in the software industry.
  8. **Documents software assets:** UML documentation removes the company from dependence on key personnel who may leave at any time and improves understanding of the company's critical business processes.
  9. **Universal:** UML is a universal language because it can be applied in many areas of software development.

3.3 DIAGRAMS IN UML

- A diagram is the graphical presentation of a set of elements, most often reduced as a connected graph of vertices (things) and arcs/paths (relationships).
- We draw diagrams to visualize a system from different perspectives so a diagram is a projection into a system.
- UML diagrams are broadly categorized as Structural and Behavioral diagrams as shown in Fig. 3.1.

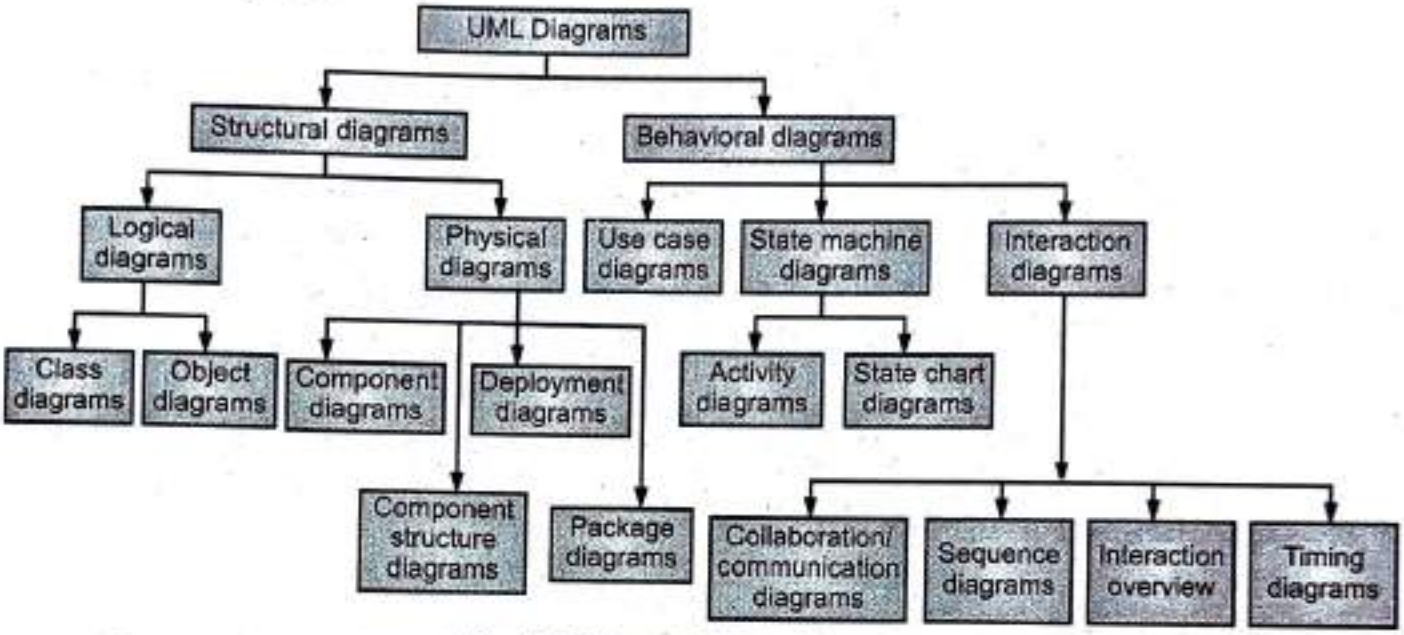


Fig. 3.1: Types of UML Diagrams

Structural Diagrams:

- This is a type of diagram that describes the elements of a specification those are irrespective of time. These diagrams show the things in a system being modeled.
- The structural diagrams are further classified as:
  1. **Logical diagrams** represent the logical structure of the system.
  2. **Physical diagrams** represent the physical structure of the system.
  3. **Behavioral Diagrams:** This is a type of diagram that describes behavioral features of a system or business process. These diagrams show what should happen in a system. They describe how the objects interact with each other to create a functioning system.
- The behavioral diagrams can be classified as Use Case diagrams, State Machine diagrams and Interaction diagrams. Interaction diagrams include sequence and collaboration diagrams, whereas state machine diagrams include state charts and activity diagrams.
- To view a system in different perspective, UML included nine different types of diagrams as follows:
  1. **Class Diagram:** It shows a set of classes, interfaces and collaborations and their relationships. Graphically, a class diagram is a collection of vertices and arcs. A class diagram addresses the static design view of a system.

2. **Object Diagram:** An object diagram shows a set of objects and their relationships at a point in time. Object diagram represents static snapshots of instances of things found in class diagrams. An object diagram addresses the static design view of a system.
3. **A Use Case Diagram:** A use case diagram represents or shows a set of use cases and actors and their relationships. Use case diagrams address static views of a system.
4. **Interaction Diagram:** An interaction diagram shows an interaction of a set of objects and their relationships along with messages that are shared among them. Interaction diagram address the dynamic view of a system.
5. **Collaboration Diagram:** A collaboration diagram is an interaction diagram that focuses on the structural organization of objects that send and receive messages.
6. **State Chart Diagram:** It shows a state machine which includes states, events, transitions and activities. It addresses a dynamic view of a system.
7. **Activity Diagram:** It is a special kind of state chart diagram which shows flow from one activity to another within a system. Activity diagram addresses dynamic view of a system.
8. **Component Structure Diagram:** These diagrams explore how objects cooperate to complete a task, document the internal structure of an object, or graphically describe a design structure or strategy.
9. **Deployment Diagrams:** These diagrams show configuration of runtime processing nodes and objects that exist on them.
10. **Package Diagrams:** These diagrams describe the high level organization of software packages. Package diagrams are used to divide the model into logical containers, or 'packages' and describe the interactions between them at a high level.
11. **Component Diagram:** Component diagram shows organizations and dependencies among a set of components. Component diagram addresses static implementation view of a system.
12. **Timing Diagrams:** These diagrams combine sequence and state diagrams to provide a view of an object's state over time and messages which modify that state.
13. **Sequence Diagrams:** These diagrams show the time ordering of messages and object lifelines.

3.4 CONCEPTUAL MODEL OF UML

- The vocabulary and rules of an UML language tell us how to create and read well-formed models, but they don't tell us what models you should create and when you should create them. That is the role of the software development process.

- A conceptual model in UML can be defined as "a model which is made of concepts and their relationships".
- A conceptual model is the first step before drawing an UML diagram. It helps to understand the entities in the real world and how they interact with each other.
- To model a software intensive system requires learning and understanding three major building blocks. These building blocks are the part of UML vocabulary.
- The building blocks of UML vocabulary:
  1. Things/Elements of UML.
  2. Relationships.
  3. Rules.
- Things are the basic elements in a model; while relationship binds these things together, diagrams bind collection of things.

3.5 THINGS IN UML / ELEMENTS OF UML [S-18, W-18]

- Things are the basic building blocks of the UML. We use them to write well-formed models.
- Any type of diagram has structure and it also specifies behavior.
- In UML, there are four kinds of things i.e. Structural things, Behavioral things, Grouping things and Annotational things as shown in Fig. 3.2.

Overview of things

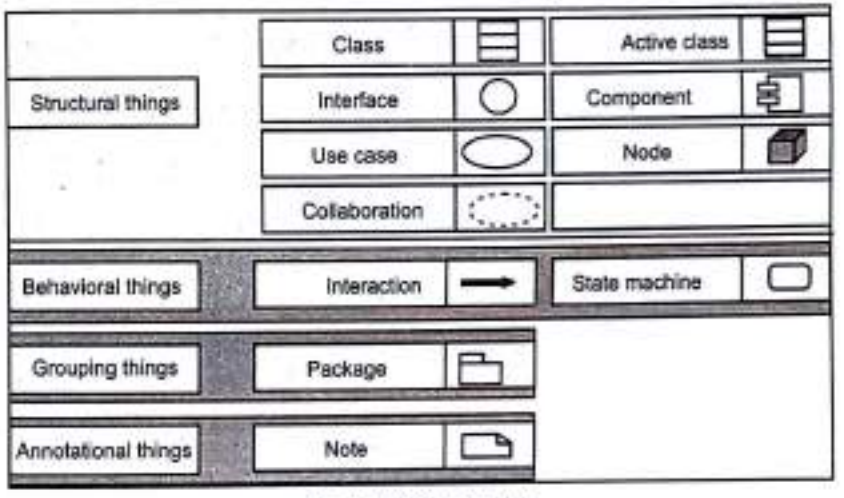


Fig. 3.2: Things in UML

Let us see details of each type of thing.

3.5.1 Structural Things

- Structural things represent a conceptual or physical element. These things are nouns of UML models. Structural things are the static parts of the model.

- Some structural things are listed below:

(i) Class:

- A class is a description of a set of objects that shares the same attributes, operation, relationship and semantics.
- A class implements one or more interfaces.
- For example, in Fig. 3.3, a class is represented with a rectangle. The rectangle includes class name, attributes and operations.

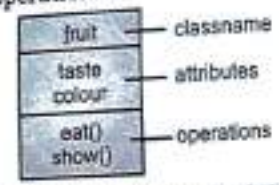


Fig. 3.3: Representation of Class

(ii) Interface:

- An interface is a collection of operations that specifies a service of a class or component.
- An interface therefore describes the externally visible behavior of that element.
- Interface defines a set of operations which specify the responsibility of a class.
- An interface might represent the complete behavior of a class or component or only a part of that behavior.
- An interface defines a set of operation specifications (that is, their sign), but never a set of operation implementations.

Declaration of interface:

- The declaration of an interface looks like a class with the keyword <<interface>> above the name; attributes are not relevant, except sometimes to show constant.
- An interface provided by a class to the outside world is shown as a small circle attached to the class box by a line.
- An interface required by a class from some other class is shown as a small semicircle attached to the class box by a line, as in Fig. 3.4.

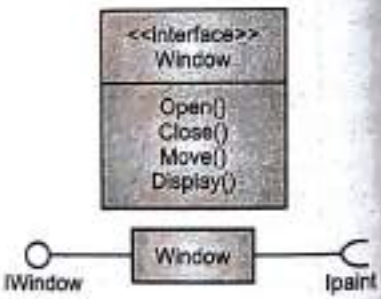
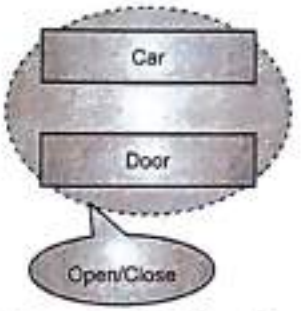


Fig. 3.4: Declaration of Interface

(iii) Collaboration:

- Collaboration defines interaction between elements. Collaboration has structural as well as behavioral dimensions. A given class or object might participate in several collaborations. In Fig. 3.5, collaborations are displayed as an ellipse with dashed lines, sometimes including only its name.

(a) Representation of Collaborations



(b) Example of Collaborations

Fig. 3.5

(iv) Use Case:

- Use case represents a set of actions performed by a system for a specific goal.
- A use case is a description of sequences of action that a system performs that return observable results of value to a particular actor.
- A use case is used to structure the behavioral things in a model. A use case is realized by collaboration.
- Graphically, a use case is displayed as an ellipse with solid lines, usually including only its name, as shown in Fig. 3.6.

Fig. 3.6: Representation of Use cases

(v) Active Class:

- An active class is a class whose objects own one or more processes or threads and therefore can initiate control activity.
- An active class is just like a class except that its objects represent elements whose behavior is concurrent with other elements.
- Graphically, it is represented as class only along with dark lines as shown in Fig. 3.7.

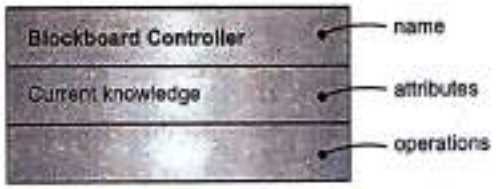


Fig. 3.7: Representation of Active class

(vi) Component:

- A component is a modular part of the system design that hides its implementation behind a set of external interfaces.

- Within a system, components sharing the same interfaces can be substituted while preserving the same logical behavior.
- The implementation of a component can be expressed by wiring together parts and connectors. The parts can include smaller components.

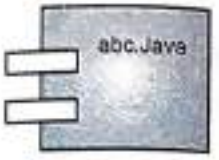


Fig. 3.8: Representation of Components

- Graphically, a component is shown at the upper right corner as shown in Fig. 3.8.

(vii) Artifacts:

- The artifacts represent physical things, whereas the previous five things represent conceptual or logical things.
- An artifact is a physical and replaceable part of a system that contains physical information.
- In a system, you will face different kinds of deployment artifacts, such as source code files, executables, script etc.
- An artifact typically represents the physical packaging of source or run-time information.
- Graphically, an artifact is as a rectangle with the keywords `<<artifact>>` above the name is shown in Fig. 3.9.

Fig. 3.9: Representation of Artifacts

(viii) Node:

- A node can be defined as a physical element that exists at run time.
- A node is a physical element that exists at run time and represents a computational resource, generally having at least some memory and, often processing capability.
- A set of components may reside on a node and may also migrate from node to node.
- Graphically, a node is shown as a cube, usually including only its name, as in Fig. 3.10.

Fig. 3.10: Representation of Nodes

3.5.2 Behavioral Things

- Behavioral things are the dynamic parts of UML models. These are the verbs of a model, representing behavior over time and space. In UML, there are three kinds of behavioral things as given below:

(i) Messages:

- Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task.
- An interaction involves a number of other elements, including messages, actions and connectors.
- Graphically, a message is displayed as a directed line, almost always including the name of its operation, as in Fig. 3.11.

Fig. 3.11: Representation of Messages

(ii) State:

- A State machine is a behavior that specifies the sequences of states an object or an interaction goes through during its lifetime in response to events.
- The behavior of an individual class or a collaboration of classes may be specified with a state machine.
- A state machine involves a number of other elements, including states, transition.
- A state is a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event.
- Graphically, a state is displayed as a rounded rectangle, usually including its name shown in Fig. 3.12.

Fig. 3.12: Representation of States

(iii) Activity:

- It is a behavior that specifies the sequences of steps a computational process performs.
- In an interaction, the focus is on the set of objects that interact. In a state machine, the focus is on the life cycle of one object at a time.
- In an activity, the focus is on the flows among steps without regard to which object performs each step. A step of an activity is called an action.
- Graphically, an action is displayed as a rounded rectangle with a name indicating its purpose. States and actions are distinguished by their different contexts.

Fig. 3.13: Representation of Action

3.5.3 Grouping Things

- Grouping things are the organizational parts of UML models. These are the boxes into which a model can be decomposed.
- Grouping things can be defined as a mechanism to group elements of an UML model together.

(i) Package:

- Package is the only one grouping thing available for gathering structural and behavioral things.
- A package is a general purpose mechanism for organizing the design itself, as opposed to classes which organize implementation constructs.
- Structural things, behavioral things and even other grouping things may be placed in a package.
- Graphically, a package is displayed as a tabbed folder, usually including only its name and sometimes its contents, as shown in Fig. 3.14.

Fig. 3.14: Representation of Packages

3.5.4 Annotational Things

- Annotational things are the explanatory parts of UML models.
- Annotational things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements.
- Note is the only one Annotational thing available.

(i) Note:

- A note is used to show comments, constraints etc. of an UML element.
- A note is simply a symbol for displaying constraints and comments attached to an element or a collection of elements.
- Graphically, a Note is displayed as a rectangle with a dog eared corner together with textual or graphical comments, as shown in Fig. 3.15.

Fig. 3.15: Representation of Note

3.6 RELATIONSHIPS IN UML [S-18, W-18]

- Relationship is another most important building block of UML.
- A relationship is a connection between things. When we model a system, we are supposed to not only identify the things that form the vocabulary of the system, but we should model how these things are related to one another.
- Relationship shows how elements are associated with each other and this association (linked) describes the functionality of an application.
- There are four kinds of relationships in the UML, i.e., Dependency, Association, Generalization and Realization. These relationships are explained below:

1. Dependency:

- Dependency is a semantic relationship. Dependency is a relationship between two things in which change in one element also affects the other one.
- Dependency states that a change in specification of one thing (independent thing) may affect another thing (depending thing) that uses it, but not the reverse.
- Graphically, dependency relationship is represented with a dashed line with an arrow, (See Fig. 3.16).

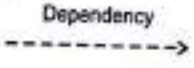


Fig. 3.16: Representation of Dependency

- For example, in the Fig. 3.17, the **FlimClip** class is depending on the **Channel** class. The dependency relationship is shown using the dashed or dotted line starting from FilmClip class and ending in the Channel class with an arrow pointing to it.

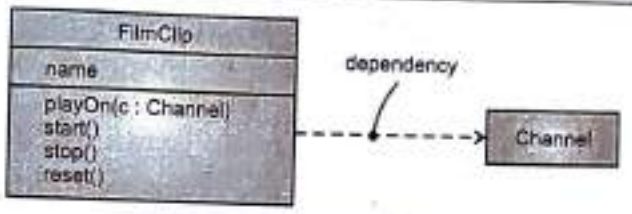


Fig. 3.17: Example of Dependency

2. Association:

- Association is a structural relationship in which objects are dually dependent.
- Association relationship is used when we want to show structural relationship.
- Association is basically a set of links that connects elements of an UML model. It also describes how many objects are taking part in that relationship.
- Graphically, association is represented as a solid line. For example, Fig. 3.18 shows linking of objects Airplane and Passengers.

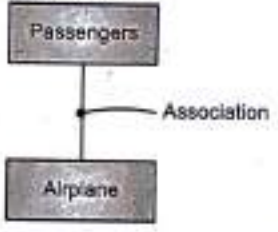


Fig. 3.18: Example of Association

Types of Association:

- There are two types of Association:

1. Aggregation:

- It refers to the formation of a particular class as a result of one class being aggregated or built as a collection.
- For example, the class "Library" is made up of one or more books, among other materials.
- In aggregation, the contained classes are not strongly dependent on the life cycle of the container.
- In the example of Fig. 3.19. Books will remain so even when the Library is dissolved.
- To show aggregation graphically in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.

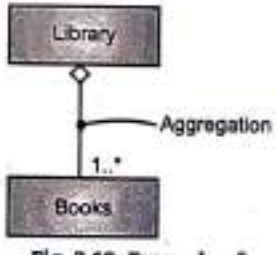


Fig. 3.19: Example of Aggregation

2. Composition:

- It is very similar to the aggregation relationship, with the only difference being its key purpose of emphasizing the dependence of the contained class to the life cycle of the container class.

- That is, the contained class will be destroyed when the container class is destroyed. For example, House (parent) and Room (child). Room doesn't exist separate from the House.
- To describe a composition relationship in an UML diagram, use a directional line connecting the two classes, with a filled diamond shape (See Fig. 3.20), adjacent to the container class and the directional arrow to the contained class.

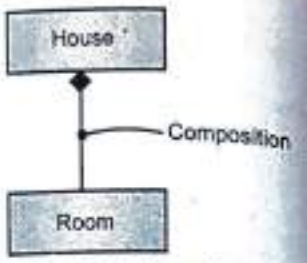


Fig. 3.20: Example of Composition Relationship

3. Generalization:

- A generalization is a relationship between a general thing i.e. super class or parent class and a specific thing i.e. subclass or child class. It is similar to the concept of inheritance in OOPs.
- Generalization can be defined as "a relationship which connects a specialized element with a generalized element."
- It basically describes inheritance relationships in the world of objects.
- It is also known as an **"is a"** relationship since the child class is a type of the parent class.
- Generalization is the ideal type of relationship that is used to showcase reusable elements in the class diagram. Literally, the child classes "inherit" the common functionality defined in the parent class.
- Graphically, generalization relationship is shown as a solid line with a hollow arrowhead pointing towards the parent as shown in Fig. 3.21.

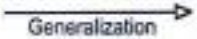


Fig. 3.21: Representation of Generalization

- Fig. 3.22 shows an example of generalization.

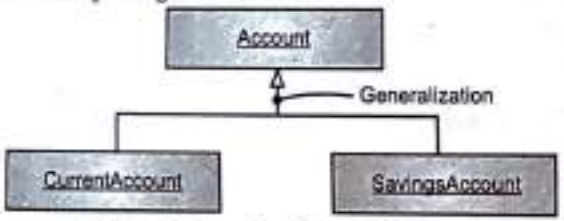


Fig. 3.22: Example of Generalization

4. Realization:

- It is a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out.

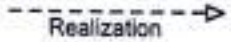


Fig. 3.23: Representation of Realization

- Realization can be defined as "a relationship in which two elements are connected".
- One element describes some responsibility which is not implemented and the other one implements them. This relationship exists in the case of interfaces.
- Graphically, a realization relationship is displayed as a cross between a generalization and a dependency relationship, as in Fig. 3.23.
- For example, in the Fig. 3.24, printing preferences that are set using the Printer setup interface are being implemented by the Printer.

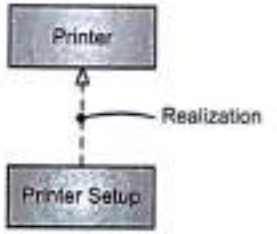


Fig. 3.24: Example of Realization

Other Important Terms used in UML:

1. Multiplicity:

- It is the active logical association when the cardinality of a class in relation to another is being depicted. It is also called Cardinality.
- For example, one fleet may include multiple airplanes, while one commercial airplane may contain zero to many passengers. The notation **0..*** in the diagram means "zero to many", (See Fig. 3.25).

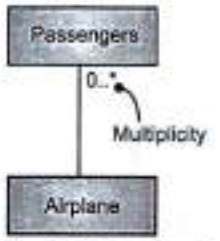


Fig. 3.25: Example of Multiplicity

3.7 RULES OF UML

- Like any language, the UML has a number of rules that specify what a well-formed model should look like.
- A well-formed model is one that is semantically self-consistent and in harmony with all its related models.
- The UML has syntactic and semantic rules for:
  1. Names : What can you call things, relationships and diagrams?
  2. Scope : The context that gives specific meaning to a name.
  3. Visibility : How can those names be seen and used by others?
  4. Integrity : How things properly and consistently related to one another?
  5. Execution : What does it mean to run or simulate a dynamic model?
- Models built during the development of a software-intensive system tend to evolve and may be viewed by many stakeholders in different ways and at different times.
- For this reason, it is common for the development team to not only build models that are well-formed, but also to build models that are:
  - Omitted : Certain elements are hidden to simplify the view.
  - Incomplete : Certain elements may be missing.
  - Inconsistent : The integrity of the model is not guaranteed.

3.8 UML ARCHITECTURE [S-19]

- Any real-world system is used by different users like developers, testers, business people, analysts and many more. So before designing a system the architecture is made with different perspectives in mind.
- The most important part is to visualize the system from a different reviewer's perspective. The better we understand the better we make the system.
- UML plays an important role in defining different perspectives of a system. These perspectives are Logical/Design, Implementation, Process and Deployment and Use Case View.

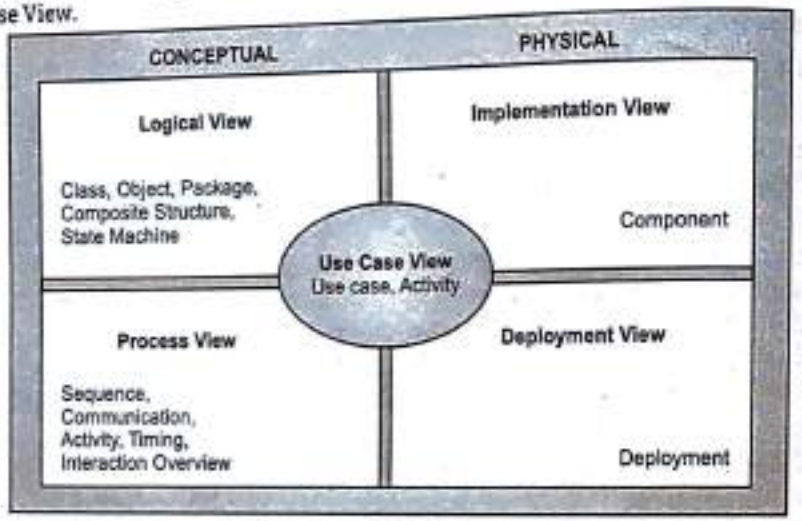


Fig. 3.26: UML Architecture

1. **The Use Case View** of a system includes the use cases that describe the behavior of the system as seen by its end users, analysts and testers.

   This view does not really specify the organization of a software system. Rather, it exists to specify the forces that shape the system's architecture.

   With the UML, the static aspects of this view are captured in use case diagrams; the dynamic aspects of this view are captured in interaction diagrams, state diagrams and activity diagrams.
2. **The Logical/Design View** of a system includes the classes, interfaces and collaborations that form the vocabulary of the problem and its solution. This view primarily supports the functional requirements of the system, meaning the services that the system should provide to its end users. With the UML, the static aspects of this view are captured in class diagrams and object diagrams, the dynamic aspects of this view are captured in interaction diagrams, state diagrams and activity diagrams. The internal structure diagram of a class is particularly useful.

3. **The Process/Interaction View** of a system shows the flow of control among its various parts, including possible concurrency and synchronization mechanisms. This view primarily addresses the performance, scalability and throughput of the system. With the UML, the static and dynamic features of this view are captured in the same kinds of diagrams as for the design view, but with a focus on the active classes that control the system and the messages that flow between them.
4. **The Implementation View** of a system includes the artifacts that are used to assemble and release the physical system. This view primarily addresses the configuration management of the system's releases, made up of somewhat independent files that can be assembled in various ways to produce a running system. It is also concerned with the mapping from logical classes and components to physical artifacts. With the UML, the static aspects of this view are captured in artifact diagrams; the dynamic aspects of this view are captured in interaction diagrams, state diagrams and activity diagrams.
5. **The Deployment View** of a system includes the nodes that form the system's hardware topology on which the system executes. This view primarily addresses the distribution, delivery and installation of the parts that make up the physical system. With the UML, the static aspects of this view are captured in interaction diagrams, state diagrams and activity diagrams.

3.9 ADVANTAGES OF UML

- Various advantages of UML are listed below:

1. **It is a formal language:** Each element has a strongly defined meaning. When we model a system, we can model with confidence that the designed model is understood by everyone.
2. **It is concise:** The entire language is made up of simple and straightforward notation.
3. **It is comprehensive:** It describes all important aspects of a system.
4. **It is scalable:** Wherever needed, the language is formal enough to handle massive system modeling projects, but it also scales low to small projects, avoiding overload.
5. **It is built on lessons learned:** UML is the result of best practices in object-oriented communities over the past few years.
6. **It is standard:** UML is controlled by an open standard group with active contributions from a world-wide group of vendors and academics.
7. **Mostly used:** It is the most useful method of visualization and documenting software system design.
8. **It is independent:** It uses object-oriented design concepts and it is independent of programming language.

3.10 DISADVANTAGES OF UML

- Disadvantages of UML are listed below:

1. **Lack of formality:** UML has still no structure and specification for modeling user interfaces. UML does not define a standard file format.
2. **Time:** Some developers might find when using UML is the time it takes to manage and maintain UML diagrams. To work properly, UML diagrams must be synchronized with the software code, which requires time to set up and maintain, and adds work to a software development project.
3. **Diagrams can get overcomplicated:** When creating an UML diagram in conjunction with software development, the diagram might become over-complicated, which can be confusing and frustrating for developers.
4. **Too much importance on design:** UML places much importance on design, which can be problematic for some developers and companies.
5. **Synchronizing code with models is difficult:** Using multiple models/diagrams makes it difficult to keep them consistent with each other and much code has to be added by hand.

Summary

- UML stands for Unified Modeling Language. The UML is a standard graphical language for modeling object-oriented software.
- UML can be defined as "UML is a language for visualizing, specifying, constructing and documenting artifacts of a software intensive system".
- A diagram is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and arcs/paths (relationships).
- We draw diagrams to visualize a system from different perspectives so a diagram is a projection into the system.
- UML diagrams are broadly categorized as Structural and Behavioral diagrams.
- Structural Diagram is a type of diagram that describes the elements of a specification those are irrespective of time.
- Behavioral Diagram is a type of diagram that describes behavioral features of a system or business process. These diagrams show what should happen in a system. They describe how the objects interact with each other to create a functioning system.
- To view a system in a different perspective, UML included nine different types of diagrams.
- The UML vocabulary has three kinds of building blocks: Things/Elements of UML, Relationships and Rules.

- Things are the basic elements in a model, while relationships bind these things together, diagrams bind collections of things.
- In UML, there are four kinds of things i.e. Structural things, Behavioral things, Grouping things and Annotational things.
- Relationship shows how elements are associated with each other and this association (linked) describes the functionality of an application.
- There are four kinds of relationships in the UML, i.e., Dependency, Association, Generalization and Realization.
- UML plays an important role in defining different perspectives of a system. These perspectives are Logical Design, Implementation, Process and Deployment and Use Case View.

Check Your Understanding

1. UML stands for _____.
   (a) Unified middle language (b) Unified modeling language
   (c) Uniform middle language (d) Uniform modeling language
2. "UML is a language for visualizing, specifying, _____ and documenting artifacts of a software intensive system".
   (a) constructing (b) contracting
   (c) contributing (d) conduction
3. The structural diagrams are further classified as: Logical diagrams, _____ diagrams.
   (a) Physical (b) Structural
   (c) Graphical (d) None
4. _____ diagram shows a set of classes, interfaces and collaborations and their relationships.
   (a) Object (b) Class
   (c) Collaboration (d) Sequence
5. There are four kinds of relationships in the UML, i.e., Dependency, _____, Generalization and Realization.
   (a) Association (b) Negotiation
   (c) Union (d) Subset
6. Activity diagram, use case diagram, collaboration diagram, and sequence diagram are _____ type of diagrams.
   (a) Non-behavioral (b) Non-structural
   (c) Behavioural (d) Structural

7. Class diagram, component diagram, object diagram, and deployment diagram are _____ diagram.
   (a) Structural (b) Behavioural
   (c) Non-behavioral (d) Non structural
8. _____ diagrams are used to show distribute files, libraries, and tables across topology of the hardware.
   (a) Deployment (b) Use case
   (c) Sequence (d) Collaboration
9. A _____ is a relationship between a general thing i.e. super class or parent class and a specific thing i.e. subclass or child class.
   (a) Generalization (b) Realization
   (c) Association (d) Dependency
10. _____ can be defined as "a relationship in which two elements are connected".
    (a) Generalization (b) Realization
    (c) Association (d) Dependency
11. There are two types of Association: Aggregation and _____.
    (a) Generalization (b) Realization
    (c) composition (d) Dependency
12. _____ is a relationship between two things in which change in one element also affects the other one.
    (a) Generalization (b) Realization
    (c) composition (d) Dependency

Answers

1. (b)	2. (a)	3. (a)	4. (b)	5. (a)	6. (c)	7. (a)	8. (a)	9. (a)	10. (b)
11. (c)	12. (d)								

Practice Questions

Q.I Answer the following questions in short.

1. What is UML?
2. Which are the types of Structural things?
3. Which are the types of Behavioral things?
4. What is an artifact?
5. How many types of relationships are in UML?
6. Write short note on following:
   (i) Structural things
   (ii) Behavioral things
   (iii) Grouping things
   (iv) Annotational things

   (v) Generalization
   (vi) Realization
   (vii) Association
   (viii) Dependency
   (ix) Package
   (x) Node
   (xi) Use Case
   (xii) Collaboration
   (xiii) Component
   (xiv) Active Class

Q.II Answer the following questions.

1. Explain advantages and disadvantages of UML.
2. Define things. Explain any one thing is UML with an example.
3. Explain UML architecture with a well labelled diagram.
4. List and explain relationships in UML with an example.
5. Explain all structural things with well labelled diagram.
6. List and explain in short about various types of diagrams in UML.

Q.III Define the terms.

1. UML
2. Relationship
3. Modeling
4. Association
5. Interface
6. Class
7. Note

Previous Exam Questions

Summer 2018

1. Define UML. What are the goals of UML? **[4 M]**
Ans. Refer to Sections 3.1 and 3.1.1.

2. What is Association? Explain important terms in Association. **[4 M]**
Ans. Refer to Section 3.6.

3. Define things. Explain Behavioral things in details. **[4 M]**
Ans. Refer to Section 3.5.2.

4. Define the term: Note **[1 M]**
Ans. Refer to Section 3.5.1.

Winter 2018

1. Define association. **[2 M]**
Ans. Refer to Section 3.6.

2. Define UML. Explain various features of UML. [4 M]

Ans. Refer to Section 3.1.

3. Define Relationship. Explain different kinds of relationship. [4 M]

Ans. Refer to Section 3.6.

4. Explain the concept of Aggregation with an example. [4 M]

Ans. Refer to Section 3.6.

Summer 2019

1. Define UML. Explain architecture of UML. [4 M]

Ans. Refer to Sections 3.1 and 3.8.

2. Define thing. Explain type of things in UML. [4 M]

Ans. Refer to Section 3.5.

■■■

4... Object Oriented Concepts and Principles

Learning Objectives ...

- To know about the concept of Object Orientation.
- To get information about Object Oriented Software development.
- To learn how to identify the elements of an object model.
- To know how to identify classes and objects.
- To study how to specify attributes.

4.1 WHAT IS OBJECT ORIENTATION?- INTRODUCTION [S-18, W-18]

- In Object-oriented, the term "object" is about the most general word in the English language and we can define it as "a thing presented to or capable of being presented to the senses".

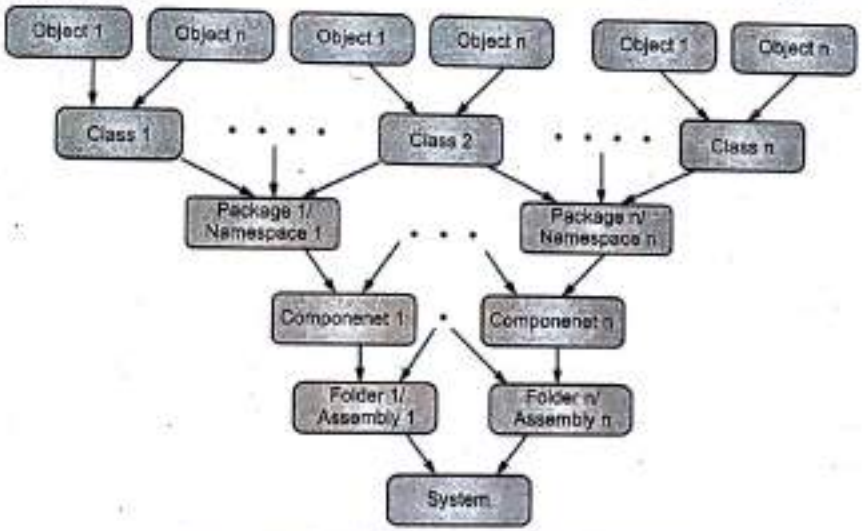


Fig. 4.1: A Typical View of Object-Oriented Approach

4...

Object Oriented Concepts and Principles

Learning Objectives ...

- To know about the concept of Object Orientation.
- To get information about Object Oriented Software development.
- To learn how to identify the elements of an object model.
- To know how to identify classes and objects.
- To study how to specify attributes.

4.1 WHAT IS OBJECT ORIENTATION?- INTRODUCTION [S-18, W-18]

- In Object-oriented, the term "object" is about the most general word in the English language and we can define it as "a thing presented to or capable of being presented to the senses".

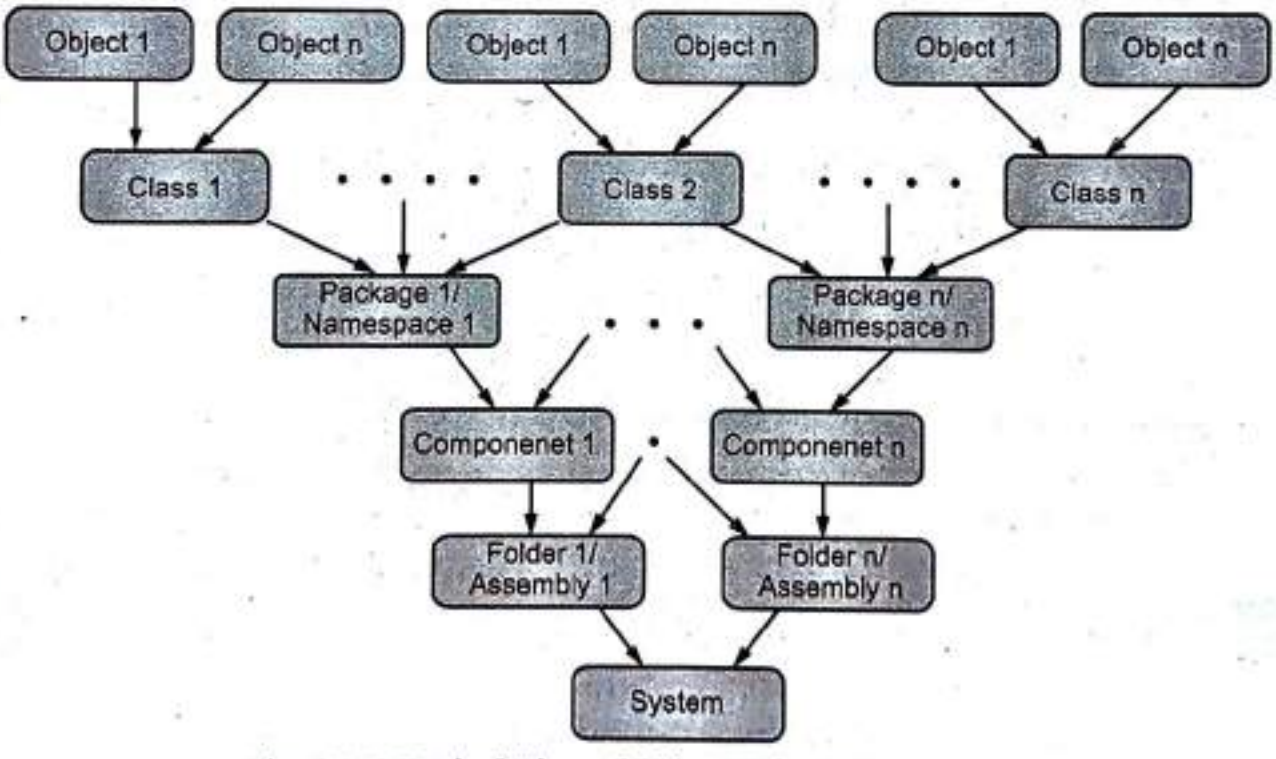


Fig. 4.1: A Typical View of Object-Oriented Approach

- An Object orientation is a technique for system modeling by considering object as a basic building block. System modeling helps the analyst to understand the functionality of the system.
- An Object is an instance of a class. All the instances/objects have same state and behavior in a given class.

  Example: If Student is a class, then specific student from that class can be the object/instance of that class.
- Fig. 4.1 shows a typical view of object-oriented approach.
- In an Object-Oriented environment:
  1. Software is a collection of discrete objects that encapsulate their data as well as the functionality to model real-world objects.
  2. An object-orientation yields important benefits to the practice of software construction.
  3. Each object has attributes (data) and methods (functions).
  4. Objects are grouped into classes. In object-oriented terms, we discover and describe the classes involved in the problem domain.

Principles of Object-Orientation (OO):

- Following are the major principles of Object-Orientation model:
  1. **Abstraction:** It is the process which allows us to focus on the most important aspects overlooking less important details.
  2. **Modularity:** It allows us to break-up complex, critical systems into small, self-contained parts that can be managed independently.
  3. **Hierarchy:** It is a tree-like structure and used to order the abstraction. This is also known as Inheritance.

     There are different types of inheritance,
     (i) Single Inheritance
     (ii) Multiple Inheritance
     (iii) Multilevel Inheritance
     (iv) Hybrid Inheritance
  4. **Encapsulation:** It mainly separates implementation from users or clients.
  5. **Polymorphism:** Polymorphism means taking more than one form. For example: In the programming, we can use + operator for performing addition of two numbers and also it is used for concatenation of two strings.

4.1.1 Reasons to use Object-Orientation [S-18]

- Various reasons to use object-orientation are given below:
  1. **Higher level of abstraction:** The object-oriented approach or technique supports abstraction at the object level. Since objects encapsulate both data (attributes) and functions (methods), they work at a higher level of abstraction. The development

can proceed at the object level and ignore the rest of the system for as long as necessary. This makes designing, coding, testing, and maintaining the system much simpler and easier.

2. **Continuous transition among different phases of software development:** The traditional approach or technique to software development requires different styles and methodologies for each step of the process. Moving from one phase to another requires a complex transition of perspective between models that can be in different worlds. This transition not only slows down the development process but also increases the size of the project. The chance for errors is also introduced in moving from one language to another. The object-oriented approach, on the other hand, essentially uses the same language to talk about analysis, design, programming, and database design. This unified approach or technique reduces the level of complexity and redundancy and makes for clearer, more robust system development.
3. **Encouragement of good programming techniques:** A class in an object-oriented system carefully describes between its interfaces, the routines and attributes within a class are held together tightly. In a properly designed system, the classes will be grouped into subsystems but remain independent; therefore, changing one class has no impact on other classes. And so, the impact is minimized.
4. **Promotion of reusability:** Objects are reusable because they are modeled directly out of a real-world problem domain. Each object stands by itself or within a small circle of peers (other objects). Within this framework, the class does not concern itself with the rest of the system or how it is going to be used within a particular system.
5. **Easy to adapt and maintain:** The systems are easier to adapt to changing requirements, easier to maintain, more robust. It promotes greater design and code reuse.
6. **Better model for problem domain:** Object-oriented methods enable to create sets of objects that work together to produce better software models than similar systems produced by traditional techniques.

4.1.2 Object-Orientation Terminology

- There are many terms available for the object-oriented concepts we have seen and different people use different terms to refer to the same things.
- Table 4.1 lists some common terms used in Object Orientation (OO):

Table 4.1: OO Terminologies

Terms	OO Terminology
Object	Objects are separate, distinguish and basic run-time entities.
Classes	A class is a collection of objects of similar type.
Attribute	It is a small piece of information like color, height etc., that describes one characteristic of an object.

contd. ...

Field	It is a named value inside an object.
Operation	It is a piece of code belonging to an object.
Method	It is a synonym for operation.
Instance	An object is an instance of a class.
Message	It is a request sent from one object to another.
Invocation	The carrying out of an operation in response to a message.
Execution	It is a synonym for invocation.
Association	It is a direct or indirect connection between two objects.
Aggregation	It is a strong association implying some kind of part of the whole hierarchy.
Composition	It is a strong aggregation where the part is inside exactly one whole. The part may also be created and destroyed by the whole.
Interface	It is a set of messages understood by an object.
Protocol	It is an agreed way of passing messages over a network.
Behavior	It is a collective term for all of an object's operation.

4.1.3 Object

- Object is the basic unit of object-orientation approach.
- An object is a real-world element in an object–oriented environment that may have a physical or a conceptual existence.
- An object is a piece of structured data in a running software system.
- Each object has:
  1. Identity that distinguishes it from other objects in the system.
  2. State that determines the characteristic properties of an object as well as the values of the properties that the object holds.
  3. Behavior that represents externally visible activities performed by an object in terms of changes in its state.
- Object can be defined technically as "a combination of data and their associated functions."

OR

- An object is "an abstraction of something of interest to the program, normally something in the real world".
- An object-oriented model consists of a number of objects; these are clearly delimited parts of the modeled system.
- Any real-world thing can be treated as an object in the OO as shown in Fig. 4.2.

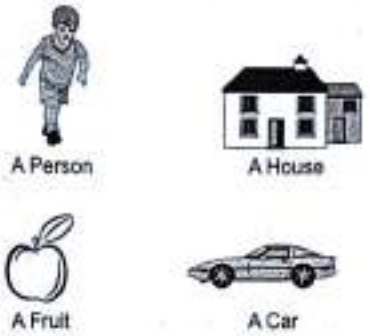


Fig. 4.2: Example of Objects

Object and Operation:

- Operations are the methods that abstract services of objects. An operation is a function that may be applied to or by object in a class.
- An example of information which we want to save for each object is name, class, percentage and so on. To access the saved information or to change the saved information, define a set of operations like accept, display etc.

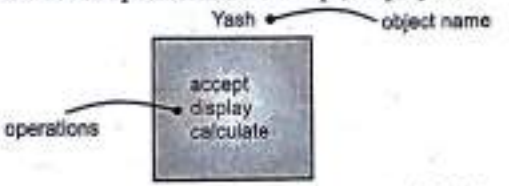


Fig. 4.3: Object Yash and Operations (activities)

- In the Fig. 4.3, an object 'Yash' is represented from outside. We are able to see operations, only the properties are hidden in an object oriented system.
- In Object Oriented Modeling, objects have relations with one another. There are two types of relations, Static Relation and Dynamic Relation.
  1. **Static Relations** are the relations existing over a long period which means that two objects know existence of each other.
  2. **Dynamic Relations** are the relations by which two objects actually communicate with each other.

4.1.4 Classes and Instances

Classes:

- A class represents a template (specification) for several objects and describes how these objects are structured internally.
- The objects that have similar behavior and information structure can be grouped under a class.
- A class serves as a blueprint for its objects. That is, once a class has been defined, any number of objects belonging to that class can be created.

- A class is defined as "a user-defined data type which contains the entire set of a similar data and the functions that the objects possess".

OR

- Class is defined as "an abstract data type characterized by a set of properties (attributes and functions) common to its objects".
- Fig. 4.4 shows an example of class. The data abstractions (attributes) that describe the class are enclosed by a "wall" of procedural abstractions (called operations, methods, or services) that are capable of manipulating the data in some way.

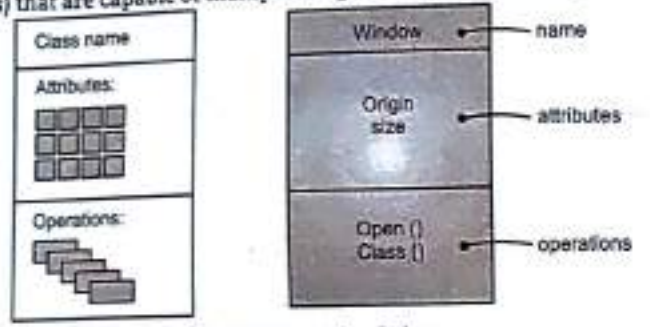


Fig. 4.4: Example of Class

Instance:

- An instance of a class is an actual object of that class. In other words, an object is an instance or occurrence of a class.
- The class is the general, abstract representation of an object, and an instance is its concrete representation.
- The objects of a class are also known as the instances or the variables of that class and the process of creating objects from a class is known as Instantiation.

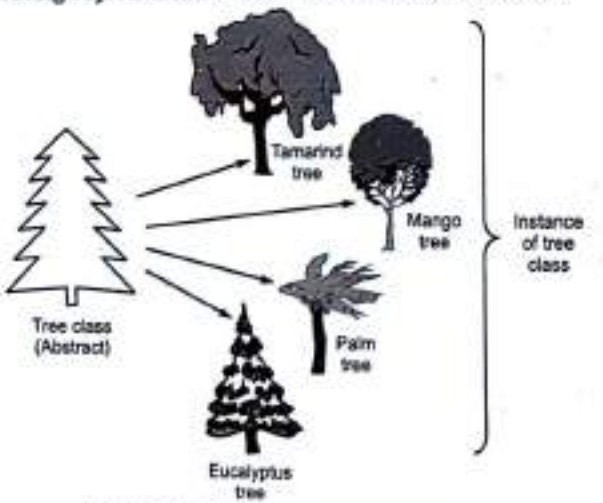


Fig. 4.5: The Tree Class and Instances

- The class describes the (behavior and information) structure of the instance, while the current state of the instance is defined by the operations performed on the instance.
- We can thus define a class Tree and each object that represents a tree becomes an instance of this class. In Fig. 4.5 we can describe a class Tree where Palm Tree, Mango Tree, Tamarind Tree and Eucalyptus Tree are instances of this class.
- The behavior of the instance and its information structure are, thus defined by its class. Each instance also has a unique identity i.e., name.
- A Tree instance and a Tree object are the same thing.

4.1.5 Polymorphism

[S-18, S-19]

- Polymorphism is a Greek term in which poly means "many" and morph means "form". It means the ability to take more than one form.
- System behavior is performed when instances start to communicate with each other. An instance may know the existence of another instance to which message has to be sent.
- In an object oriented system, polymorphism means sending an instance does not need to know the receiving instance's class. The receiving instance can belong to any class.
- Polymorphism means different instances can be associated and these instances can belong to different classes.
- Polymorphism is a very important characteristic for modeling. It is a strong tool for developing flexible systems. If we add an object of a new class, this modification will affect only the new object and not those sending messages to it.
- Grady Booch defines Polymorphism as "the relationship of objects of many different classes by some common super class; thus, any of the objects designated by this name is able to respond to some common set of operations in a different way".
- Polymorphism plays an important role in allowing the objects having different internal structures to share the same external interface.
- In other words, this means that a general class of operation may be accessed in the same manner although space action associated with each operation may differ.
- Fig. 4.6 shows an example of polymorphism. In this fig., Shape is a class which contain Draw() method, which takes many forms like Draw (circle), Draw (rectangle) etc.

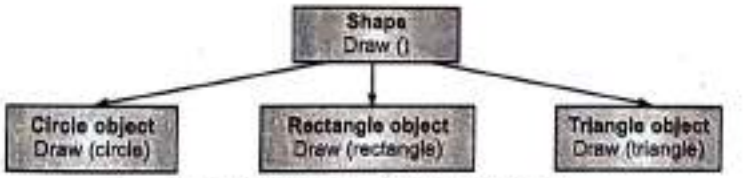


Fig. 4.6: Example of Polymorphism

[S-18]

4.1.6 Inheritance

- The process by which objects of one class acquire the properties of objects of another class is called as Inheritance.
- Inheritance is the process of forming a new class from an existing class or base class. The base class is also known as Parent Class or Super Class.
- The new class that is formed is called derived class. Derived class is also known as a Child Class or Subclass.
- Inheritance helps in reducing the overall code size of the program which is an important concept in Object Oriented Programming.
- In Fig. 4.7, the Santro and Accent are a part of the class Hyundai which is again part of the class Car and Car is the part of the class Vehicle. That means Vehicle class is the parent class.

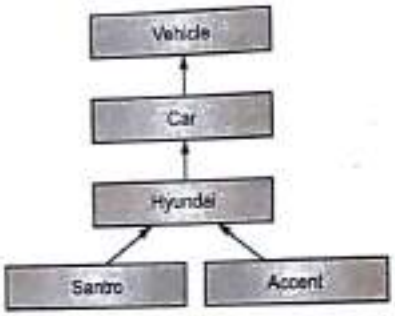


Fig. 4.7: Example of Inheritance

- There are five types of inheritance. Let us see details each of them.

1. **Single Inheritance:** A derived class with only one base class is called 'Single Inheritance'. In Fig. 4.8 (a), A is a base class and B is a derived class.

Fig. 4.8 (a): Single Inheritance

2. **Multiple Inheritance:** A derived class with more than one base class is called Multiple Inheritance. In Fig. 4.8 (b), class A and class B are base classes and class C is a derived class.

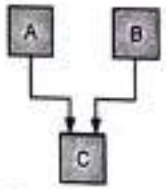


Fig. 4.8 (b): Multiple Inheritance

3. **Multilevel Inheritance:** When a new class is derived from another derived class, it is called as Multilevel Inheritance. We can inherit new classes until any level.

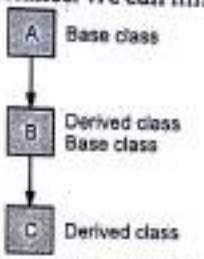


Fig. 4.8 (c): Multilevel Inheritance

In Fig. 4.8 (c), A is a base class, B is derived from class A. class B is a derived class and play role of base class from which the class C is derived.

4. **Hierarchical Inheritance:** Any number of new classes can be derived from a single base class. It is called as Hierarchical Inheritance.

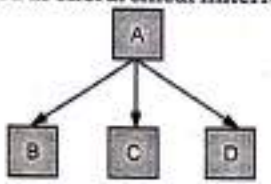


Fig. 4.8 (d): Hierarchical Inheritance

In Fig. 4.8 (d), classes B, C and D are derived from the single base class A.

5. **Hybrid Inheritance:** Two or more types of inheritances are combined to design a program. Class A is a base class, class B is derived from A and C is derived from base classes B and D. Two types of inheritances are combined in below inheritance example: (1) Single, (2) Multiple.

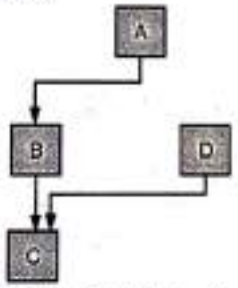


Fig. 4.8 (e): Hybrid Inheritance

4.2 OBJECT ORIENTED SYSTEM DEVELOPMENT

- Development refers to the software life cycle i.e. analysis, design and implementation.
- System development process is considered as a sequence of well defined activities done throughout its life cycle.

- Object oriented development approach encourages software developers to work and think in terms of the application throughout the software life cycle.
- OO development is a conceptual process independent of programming language until the final stage.
- Object oriented development offers a different model from the traditional software development approach, which is based on functions and procedures.
- In simplified terms, object oriented systems development is a way to develop software by building self-contained modules or objects that can be easily replaced, modified, and reused.

4.2.1 Introduction

- System development is the process of defining, designing, testing, and implementing a new system.
- Fig. 4.9 shows the system development process.

Fig. 4.9: System Development Process

- The term 'Systems development' refers to all activities that go into producing an information systems solution.
- Object-oriented systems development methods differ from traditional development techniques.
- Traditional techniques view software as a collection of programs (or functions) and isolated data.
- A program can be defined as,
  Algorithms + Data Structures = Programs.
- A software system is a set of mechanisms for performing certain action on certain data.
- In object oriented a program is simply a collection of well designed objects.
- A program in object-oriented can be defined as,
  Data + Operations = Object
- It is shown in Fig. 4.10.

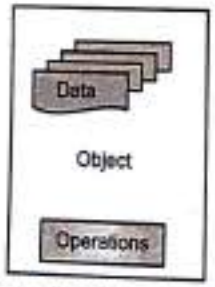


Fig. 4.10: Concept object oriented program

4.2.2 Function/Data Methods (with Visibility) [W-18]

- The existing methods for system development can basically be divided into function/data methods and object-oriented methods.
- By function/data methods we mean those methods that treat functions and/or data as being more or less separate.
- Object-oriented methods view functions and data as highly integrated.
- These methods are shown schematically in Fig. 4.11.

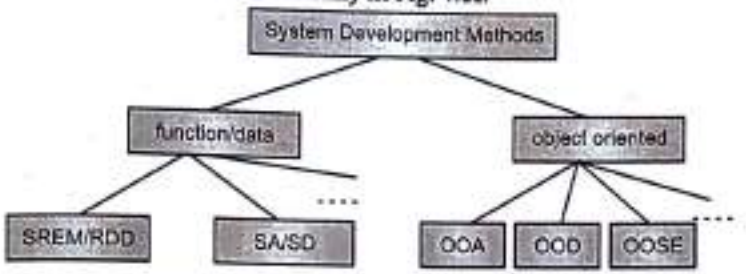


Fig. 4.11 (a): System Development Methods

- Structured Analysis and Design Technique (SADT), Requirement Driven Design (RDD) based on SREM (Software Requirements Engineering Methodology) and Structured Analysis (SA) and Structured Design (SD) are traditional software engineering function/data methods.
- Functions and data are separate functions that are active and have behaviour whereas data is a passive holder of information which is affected by functions.
- The system is divided into modules i.e. functions, while data is sent between those functions.
- It is difficult to maintain a system developed using function/data methods. A major problem with this system is that all functions must know how its data structure works.
- A small change in a developed system will affect the system, it is difficult to find the problem and correct the system. Function/Data systems become more difficult to modify.
- It is possible to use a function/data method initially and later in the design or programming phase, for object oriented programming. It is difficult to join programs and data in order to design objects and classes, a method is based on separating them.
- Object oriented programming requires a different approach from function/data methods.

Visibility:

- Visibility helps us to differentiate between the outside view and the inside view of a class. The interface of a class provides its outside view and therefore highlights the abstraction while hiding its structure and the secrets of its behaviour. This consists of

the declarations of all operations applicable to instances of this class, but it may also include the declaration of other classes, constants, variables and exceptions as needed to complete the abstraction.

- By contrast, the implementation of a class is the inside view that includes the secrets of its behaviour. It consists of the implementations of all the operations defined in the interface of the class.
- We can divide the visibility of a class in the following parts:
  - **Public:** A declaration that is accessible to all classes.
  - **Protected:** A declaration that is accessible only to the class itself and is sub classes.
  - **Private:** A declaration that is accessible only to the class itself.
  - **Package:** A declaration that is accessible only by classes in the same package.

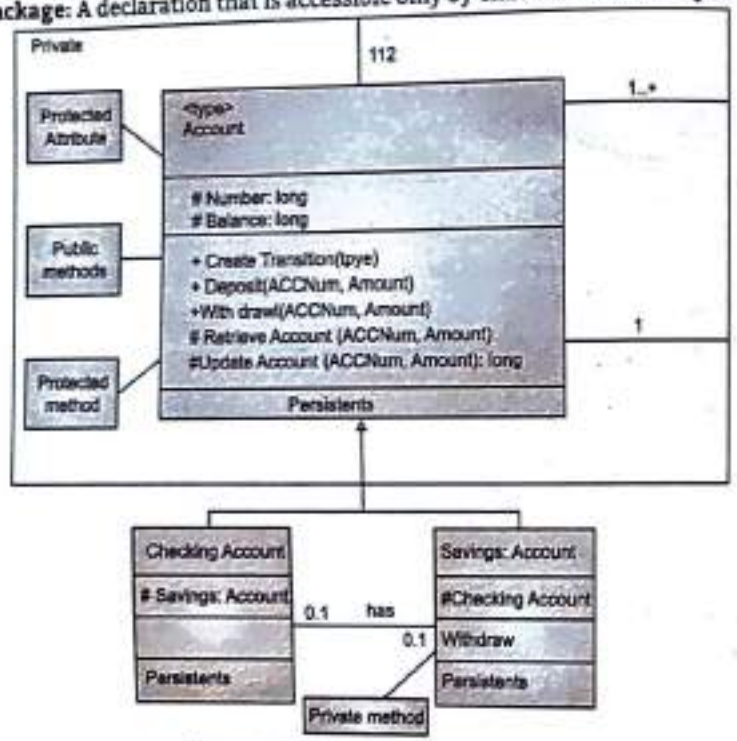


Fig: 4.11 (b): Visibility of Function/data

4.2.3 Object Oriented Analysis (OOA)

- The purpose of OOA as with all other analysis is to obtain an understanding depending only on the system's functional requirements.
- OOA is the procedure of identifying software engineering requirements and developing software specifications in terms of a software system's object model, which comprises interacting objects.

- The main difference between object-oriented analysis and other forms of analysis is that in object-oriented approach, requirements are organized around objects, which integrate both the data and functions.
- They are modelled after real-world objects that the system interacts with. In traditional analysis methodologies, the two aspects: functions and data are considered separately.
- Grady Booch has defined OOA as, "object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain".
- The primary tasks in object-oriented analysis (OOA) are:

1. **Finding the Objects:**

- Objects can be found in the application domain as naturally occurring entities. Finding objects is not difficult but selecting relevant objects for system is difficult. Objects should be selected in such a way that they will remain essential throughout system's life cycle.
- In an object oriented methods, there may be only one type of object or different types of objects. By having different types of objects, one can quickly obtain an overview of the system, while with only one type of object, it is quite difficult to see the difference between different objects.
- It is important that objects identified during analysis should be found in the design i.e. traceability. The objects may be implemented using earlier developed code i.e. components.

2. **Organizing the Objects:**

- To organize objects and classes of objects, we have to use different criteria. One of the criteria starts by considering how similar classes of objects are to each other i.e. inheritance. Another criterion can be how an object is part of another i.e. which object works with which other object.

3. **Object Interaction:**

- To get a clear idea of object's role in the system, we can use cases in which objects take part and communicate with other objects. By this, we can describe objects surroundings and exceptions of other objects.

4. **Operations on Objects:**

- The operations on objects can be identified directly from the application. The operations on objects come naturally when we consider an object interface.
- The operations can be basic operations like create, add, delete or can be difficult.

5. **Object Implementation:**

- At last, the object has to be defined which includes defining the information that each object must hold.

4.2.4 Object Oriented Construction [W-18]

- Any software development approach goes through the following stages –
  - Analysis
  - Design
  - Implementation
- The analysis model designed and implemented in source code is nothing but object oriented construction. It means the model designed in analysis phase must be modelled in the implementation environment.
- In other ways, we can say the objects identified in the analysis phase must be found within the design. It means we have to transform an analysis model into the design model and then in implementation i.e. source code by using programming language.
- The objects identified during the analysis phase may be implemented during construction phase by using previously developed source code.
- Object oriented systems can be developed by using object oriented programming.

4.3 IDENTIFYING THE ELEMENTS OF AN OBJECT MODEL [S-18, W-18]

- We already know object orientation is a technique for system modelling.
- A model is an abstract representation of a system, constructed to understand the system prior to building or modifying it.
- The object model visualizes the elements in a software application in terms of objects.
- The object model describes the structure of objects in a system: their identity, relationships to other objects, attributes, and operations.
- The object model is represented graphically with an object diagram. The object diagram contains classes interconnected by association lines. Each class represents a set of individual objects.
- Modeling is the process of describing an existing or proposed system. The purpose of modeling is to reduce complexity by building a simplified representation of reality which ignores irrelevant details.
- Using object-orientation as a base, we model the system as a number of objects that interact. Irrespective of thinking about the type of system being modeled, we should think about its contents as a number of objects which in one way or another are related.
- We have to identify the objects that are the part of our object model, which is completely dependent on which object model is to represent.
- Elements to be identified for an object model are:
  (i) Class
  (ii) Objects
  (iii) Attributes
  (iv) Operations

  Once these elements are identified, that belongs to the model to be represented.

- This is an important step which is analysis of elements to be involved in modelling.
- Objects can be found in the application domain as naturally occurring entities. Finding objects is not difficult but selecting relevant objects for system is difficult. Objects should be selected in such a way that they will remain essential throughout the systems life cycle.
- In an object oriented method, there may be only one type of object or different types of objects. By having different types of objects, one can quickly obtain an overview of the system, while with only one type of object; it is quite difficult to see the difference between different objects.
- It is important that objects identified during analysis should be found in the design i.e. traceability. The objects may be implemented using earlier developed code i.e. components.
- In reality, OOA actually attempts to define classes from which objects are instantiated. Therefore, when we isolate potential objects, we also identify members of potential classes.
- We can begin to identify objects by examining the problem statement or by performing a "grammatical parse" on the processing narrative for the system to be built.
- Objects are determined by underlining each noun or noun clause and entering it in a simple table. Synonyms should be noted.
- If the object is required to implement a solution, then it is part of the solution space; otherwise, if an object is necessary only to describe a solution, it is part of the problem space. What should we look for once all of the nouns have been isolated?
- Objects visible themselves in one of the ways represented in Fig. 4.12.

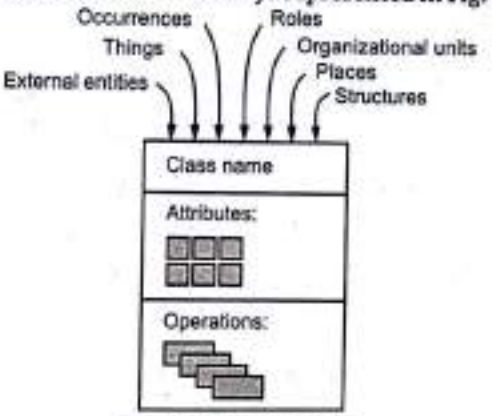


Fig. 4.12: Object Models

- Objects can be:
  1. **External entities** like other systems, devices, people etc. that produce or consume information to be used by a computer-based system.

2. **Things** like reports, displays, letters, signals etc. that are part of the information domain for the problem.
3. **Occurrences or Events** like a proper transfer or the completion of a series of robot movements that occur within the context of system operation.
4. **Roles** like manager, engineer, salesperson etc. played by people who interact with the system.
5. **Organizational units** like division, group, team etc. that are relevant to an application.
6. **Places** like manufacturing floor or loading dock that establish the context of the problem and the overall function of the system.
7. **Structures** like sensors, four-wheeled vehicles or computers etc. that define a class of objects or in the extreme, related classes or objects.

4.4 IDENTIFYING CLASSES AND OBJECTS

- The objects can be found as naturally occurring entities in the application domain.
- There is no problem in finding objects while difficulty is in selecting these objects, relevant to the system.
- The intention is to find the essential objects, which are to remain essential throughout the system's life cycle. For example, for the banking application, typical objects would be Customer, Account, Bank and Staff.
- Define the classes that model real-world entities. Generally, classes will be the nouns that figure in your problem domain like Bank, Money, Budget, Credit Card etc.
- Defining your own classes with easy to understand, class names make it much easier to keep track of your model.

4.5 SPECIFYING THE ATTRIBUTES (WITH VISIBILITY)

- Something that an object knows is called an attribute, which essentially represents data.
- A class defines attributes and an object has values for those attributes.
- An attribute is a named property of a class which describes a range of values. A class may have any number of attributes or no attributes at all.

  An attribute which has some property is shared by all objects of that class.

  Example: Customers class has a name, address, phone number and date of birth.
- You can specify the properties of the entity to be modeled along with the name of the attribute ; we can specify the default value and its datatype.
- In an abstract we specify only attribute name which is enough. But we can also specify the visibility scope and multiplicity of each attribute.
- Following is the syntax of an attribute in UML:

```
[Visibility] name [Multiplicity] [: type] [= initial value]
```

Where,

Visibility : Refers to the ability of a method to reference a function form another

Examples:

name	:	Name only
+ name	:	Visibility and name
name: string	:	Name and Type
name [0 .... 1]: String	:	Name, Multiplicity and Type
name: String = ("ABC")	:	Name, Type and Initial value

- Using Visibility, we can specify with attributes and even with operations which specifies whether it can be used by other classifiers. There are three levels of visibility in UML public, protected and private. We can specify any of these levels of visibility.
- The keywords Private, Public and Protected are considered as visibility labels. It should be noted that these visibility labels are followed by " : ".

1. **Public:** Any outside classifier with visibility to the given classifier can use the feature.

   Specified by prepending the '+' symbol.

   Example: + name
2. **Protected:** A protected member variable or function is very similar to a private member but it provides one additional benefit that they can be accessed in child classes which are called derived classes. In other words, any descendant of the classifier can use the feature.

   Specified by prepending the '#' symbol.

   Example: # phone
3. **Private:** A private member variable or function cannot be accessed, or even viewed from outside the class. Only the classifier itself can use the feature.

   Specified by prepending the '–' symbol.

   Example: –address

Fig. 4.13: Visibility of attributes

4.6 DEFINING OPERATIONS

- Something an object can do is called an operation (specification) and essentially represents processing.
- In what way an object does the processing for a given operation is known as the operation's method or implementation.

- An operation is implementation of a service that can be requested from any object of a class to affect behaviour i.e. it is an abstraction of something that you can do to an object and is shared by all objects of that class.
- Operations define the behavior of an object and change the object's attributes in some way. More specifically, an operation changes one or more attribute values that are contained within the object.
- An operation must have "knowledge" of the nature of the objects attributes and must be implemented in a manner that enables it to manipulate the data structures that have been derived from the attributes.
- A class may have any number of operations. Operations may be shown by their names.
- We can specify an operation by stating its signature, covering the name, type and default value of all parameters and a return type. We can specify visibility and scope of each operation.
- The syntax of an Operation in UML is:

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

Where,

`visibility` : + (public), # (protected), or - (private)

`name` : a string

`parameter-list` : comma-separated parameters whose syntax is similar to that for attributes: `direction name: type = default value`. The only extra element is `direction`, which is used to show whether the parameter is used for input (`in`), output (`out`), or both (`inout`). If there is no direction value, it's assumed to be in.

`return-type-expression` : a comma-separated list of return types. Most people use only one return type, but multiple return types are allowed.

`property-string` : property values that apply to the given operation.

Examples:

display	:	Name only
+ display	:	Visibility and Name
set (n: Name, s: String)	:	Name and Parameters
getID ( ): Integer	:	Name and return type

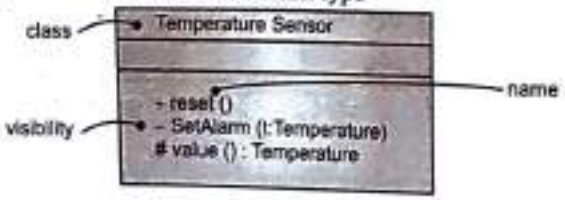


Fig. 4.14: Concept of Operations

4.7 FINALIZING THE OBJECT DEFINITION

- The intent of Object Oriented Analysis is to define all classes, their relationships and associated behaviour with them that are relevant to the problem to be solved.
- To achieve this, the following tasks will occur:
  1. Basic user requirements must be communicated between the customer and software engineer.
  2. Classes must be identified i.e. its attributes and methods are defined.
  3. A class hierarchy is defined.
  4. Object to object relationships should be represented.
  5. Object behaviour must be modeled.
  6. Above all steps reapplied repeatedly till the model is not complete.

Summary

- Object orientation is a technique for system modeling by considering an object as a basic building block. System modeling helps the analyst to understand the functionality of the system.
- Object is the basic unit of object-orientation approach.
- An object is a real-world element in an object-oriented environment that may have a physical or a conceptual existence.
- Operations are the methods that abstract services of objects. An operation is a function that may be applied to or by object in a class.
- A class is defined as "a user-defined data type which contains the entire set of similar data and the functions that the objects possess".
- Polymorphism is a Greek term in which poly means "many" and morph means "form". It means the ability to take more than one form.
- The process by which objects of one class acquire the properties of objects of another class is called as Inheritance.
- Inheritance is the process of forming a new class from an existing class or base class. The base class is also known as Parent Class or Super Class.
- The new class that is formed is called derived class. Derived class is also known as a Child Class or Subclass.
- Object oriented development approach encourages software developers to work and think in terms of the application throughout the software life cycle.
- Object-oriented methods view functions and data as highly integrated.
- OOA is the procedure of identifying software engineering requirements and developing software specifications in terms of a software system's object model, which comprises interacting objects.
- The main difference between object-oriented analysis and other forms of analysis is that in object-oriented approach, requirements are organized around objects, which integrate both data and functions.
- The analysis model designed and implemented in source code is nothing but object oriented construction. It means the model designed in analysis phase must be modeled in the implementation environment.

- The object model is represented graphically with an object diagram. The object diagram contains classes interconnected by association lines. Each class represents a set of individual objects.
- The objects can be found as naturally occurring entities in the application domain.
- A class defines attributes and an object has values for those attributes.
- An operation is implementation of a service that can be requested from any object of a class to affect behaviour i.e. it is an abstraction of something that you can do to an object and is shared by all objects of that class.

Check Your Understanding

1. Object is an instance of a _____.
   (a) class (b) attribute
   (c) structure (d) array
2. A class represents a _____ for several objects and describes how these objects are structured internally.
   (a) structure (b) template
   (c) diagram (d) interaction
3. A class is defined as "a _____ data type which contains the entire set of a similar data and the functions that the objects possess".
   (a) user-defined (b) inbuilt
   (c) derived (d) None of these
4. In an object oriented system, _____ means the sending instance does not need to know the receiving instance's class. The receiving instance can belong to any class.
   (a) Inheritance (b) Abstraction
   (c) Polymorphism (d) Message passing
5. The process by which objects of one class acquire the properties of objects of another class is called as _____.
   (a) Inheritance (b) Abstraction
   (c) Polymorphism (d) Message passing
6. Inheritance is the process of forming a new class from an existing class or base class. The new class that is formed is called _____.
   (a) Derived class (b) Super class
   (c) Parent class (d) Subclass
7. There are _____ types of inheritance.
   (a) three (b) two
   (c) seven (d) five
8. A derived class with more than one base class is called _____ Inheritance.
   (a) Single (b) Multiple
   (c) Hiercherical (d) Hybrid

9. A derived class with only one base class is called _____ Inheritance.
   (a) Single (b) Multiple
   (c) Hierarchical (d) Hybrid
10. When a new class is derived from another derived class, it is called as _____ Inheritance.
    (a) Single (b) Multilevel
    (c) Hierarchical (d) Hybrid
11. Two types of inheritances are combined; it is called as _____ inheritance.
    (a) Single (b) Multilevel
    (c) Hierarchical (d) Hybrid
12. Object-oriented system _____ is a way to develop software by building self-contained modules or objects that can be easily replaced, modified, and reused.
    (a) development (b) construction
    (c) analysis (d) design
13. _____ is the procedure of identifying software engineering requirements and developing software specifications in terms of a software system's object model, which comprises interacting objects.
    (a) OOD (b) OOA
    (c) Function (d) None of these

Answers

1. (a)	2. (b)	3. (a)	4. (c)	5. (a)	6. (a)	7. (d)	8. (b)	9. (a)	10. (b)
11. (d)	12. (a)	13. (b)							

Practice Questions

Q.I Answer the following questions in short.

1. What is Object Orientation?
2. What is Object Oriented Analysis?
3. What is Object Oriented Construction?
4. What is class? Explain with an example.
5. What is an Object? Explain with an example.
6. What is inheritance? Write types of inheritance.
7. What is polymorphism?
8. Consider a single object "Book" and draw object diagram with possible attributes.

Q.II Answer the following questions.

1. Explain object orientation. Give principles of object orientation.
2. Why to use object orientation? Explain with reasons.
3. Explain the following Object Oriented terminologies.
   (i) attribute
   (ii) field
   (iii) method
   (iv) execution

4. Define Inheritance. Explain any three types of Inheritance with well labelled diagram and an example.
5. Write short note on:
   (i) Object oriented Analysis
   (ii) Object oriented Construction
   (iii) Function/Data methods
   (iv) Polymorphism

Q.III Define the terms.

1. Object
2. Class
3. Attribute
4. Object Orientation
5. Polymorphism
6. Inheritance

Previous Exam Questions

Summer 2018

1. What is multiple inheritance? [2 M]

Ans. Refer to Section 4.1.7.

2. What is object orientation? State various reasons for why object orientation. [4 M]

Ans. Refer to Sections 4.1 and 4.1.1.

3. Define the term –Polymorphism. [1 M]

Ans. Refer to Section 4.1.6.

Winter 2018

1. What is Object Oriented Analysis? [2 M]

Ans. Refer to Section 4.4.3.

2. What is meant by Object Oriented Design? [2 M]

Ans. Refer to Section 4.4.4.

3. How to identify the element of an object model. [4 M]

Ans. Refer to Section 4.3.

4. Explain visibility modes along with well labelled diagrams. [4 M]

Ans. Refer to Section 4.2.2.

5. Discuss object oriented design process. [4 M]

Ans. Refer to Section 4.4.4.

6. What is meant by Object Oriented Analysis? [4 M]

Ans. Refer to Section 4.4.3.

Summer 2019

1. What is inheritance? [2 M]

Ans. Refer to Section Refer to Section 4.1.7.

2. Define polymorphism. [2 M]

Ans. Refer to Section 4.1.6.

■■■

5... Structural Modeling

Learning Objectives ...

- To understand Structural Modeling.
- To understand various relationships in UML.
- To draw Class diagrams and Object diagrams.
- To learn about Advanced Classes and Relationships.
- To know about Interface, Types and Roles.
- To get information about the concept of Packages.

5.1 INTRODUCTION TO STRUCTURAL MODELING

- Modeling a system involves identifying the things that are important to your particular view. These things form the vocabulary of the system that you are modeling.
- For example, if you are building a farmhouse, things like walls, doors, windows, cabinets and lights are some of the things that will be important to you as a farm home owner. Each of these things can be distinguished from the other.
- Structural modeling captures the static features of a system.
- Structural model represents the framework for the system and this framework is the place where all other components exist. So the class diagram, Object diagram, component diagram and deployment diagrams are the part of structural modeling. They all represent the elements and the mechanism to assemble them.

5.2 CLASSES [S-19]

- S-Classes are the most important building blocks of any object-oriented system.
- A class acts as a logical entity that works as a template (blueprints) for creating similar kinds of objects.
- A class is a group of objects with similar properties (attributes), common behaviour (operation), common relationship to other objects and common semantics.
- A class is a description of a set of objects that share the same attributes, operations, relationships and meaning.

5... Structural Modeling

Learning Objectives ...

- To understand Structural Modeling.
- To understand various relationships in UML.
- To draw Class diagrams and Object diagrams.
- To learn about Advanced Classes and Relationships.
- To know about Interface, Types and Roles.
- To get information about the concept of Packages.

5.1 INTRODUCTION TO STRUCTURAL MODELING

- Modeling a system involves identifying the things that are important to your particular view. These things form the vocabulary of the system that you are modeling.
- For example, if you are building a farmhouse, things like walls, doors, windows, cabinets and lights are some of the things that will be important to you as a farm home owner. Each of these things can be distinguished from the other.
- Structural modeling captures the static features of a system.
- Structural model represents the framework for the system and this framework is the place where all other components exist. So the class diagram, Object diagram, component diagram and deployment diagrams are the part of structural modeling. They all represent the elements and the mechanism to assemble them.

5.2 CLASSES [S-19]

- S-Classes are the most important building blocks of any object-oriented system.
- A class acts as a logical entity that works as a template (blueprints) for creating similar kinds of objects.
- A class is a group of objects with similar properties (attributes), common behaviour (operation), common relationship to other objects and common semantics.
- A class is a description of a set of objects that share the same attributes, operations, relationships and meaning.

- Graphically, a class is represented as a rectangle by showing its name, attributes and operations that share for all objects of that class.
- Fig. 5.1 shows the example of the Circle class. The attributes of Circle class are x-coord, y-coord, and radius. The operations are `findArea()` and `findCircumference()`.

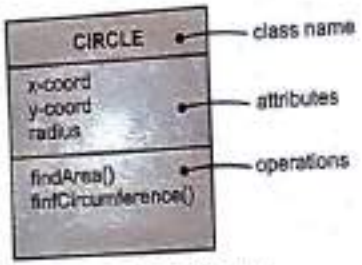


Fig. 5.1: Classes

5.2.1 Name of Class

- Every class has a name and this name distinguishes it from other classes. A name is a textual string.
- That name alone is known as a simple name and a qualified name is the class name prefixed by the name of the package in which that class lives.
- A class may be drawn showing only its name as shown in Fig. 5.2.

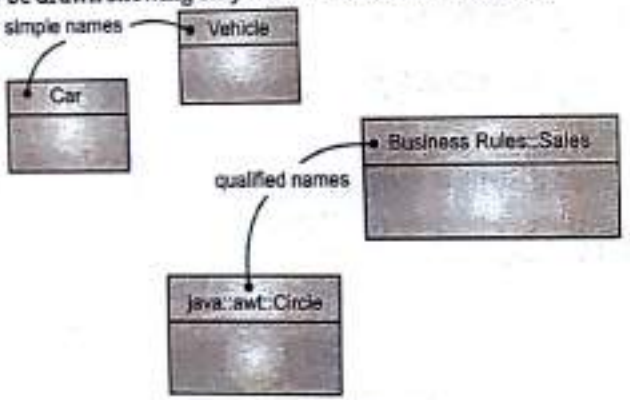


Fig. 5.2: Simple and Qualified Names

5.2.2 Object [S-18]

- Instance of a class is called as Object.
- An object has a well-defined collection of behaviours.
- **Example:** Let us consider an object of the class `Circle` named `x1`. We assume that the center of `x1` is at (2, 3) and the radius of `x1` is 5.

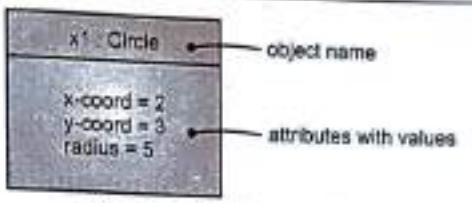


Fig. 5.3: Object of Circle Class

5.2.3 Attribute

- Something that an object knows is called an Attribute.
- An attribute is a named property of a class that describes a range of values that instances of the property may hold. A class may have any number of attributes.
- An attribute name may be text, just like a class name. In practice, an attribute name is a short noun or noun phrase that represents some property of its enclosing class.

Representation:

- Graphically, an attribute is represented in a section next to the class name, (See Fig. 5.4).

Fig. 5.4: Attributes of Employee Class

- An attribute name is a text and the first letter of the first word is in small letters and the next word's first letter can be in capital letter, i.e. name, empSalary.

5.2.4 Operations

- Something an object can do is called an Operation.
- An operation is the implementation of a service that can be requested from any object of a class to affect behavior.
- In other words, an operation is an abstraction of something you can do to an object that is shared by all objects of that class.
- A class may have any number of operations or no operations at all.
- Graphically, operations are listed in a compartment just below the class attributes. Operations may be drawn showing only their names, as Fig. 5.5.

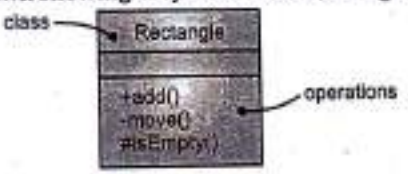


Fig. 5.5: Operations

- An operation name may be text, just like a class name. In practice, an operation name is a short verb or verb phrase that represents some behaviour of its enclosing class.

5.3 RELATIONSHIP [W-18]

- In UML, a relationship is a connection among things.
- When we model a system, we are supposed to not only identify the things that form vocabulary of system, but we should model how these things are related to one another.
- Relationship shows how elements are associated with each other and this association (linked) describes the functionality of an application.
- In Object Oriented Modeling, there are three kinds of relationships. i.e. Dependency, Generalization and Association, (See Fig. 5.6).
  - **Association** shown as a solid line with no arrowhead.
  - **Generalizations** are represented by a solid line with a solid arrowhead.
  - **Dependencies** are presented by a dotted line with an arrowhead.

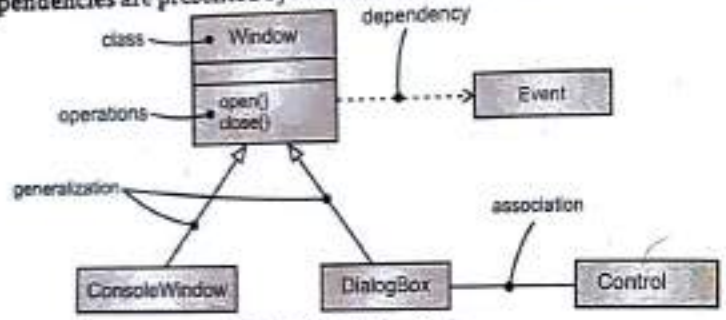


Fig. 5.6: Relationships

5.3.1 Dependency [S-18]

- Dependency is a semantic relationship. Dependency is a relationship between two things in which a change in one element also affects the other one.
- Dependency states that a change in specification of one thing (independent thing) may affect another thing (depending thing) that uses it, but not the reverse.

Representation:

- Graphically, dependency relationship is represented with a dashed line with an arrow,

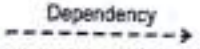


Fig. 5.7 (a): Representation of Dependency

Example:

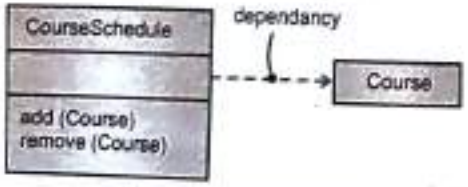


Fig. 5.7 (b): Dependency

- In the above example, dependency from `CourseSchedule` to `Course` exists because `Course` is used in both add and remove operations of `CourseSchedule`.
- Dependency is shown by pointing from a class with the operation to the class used as a parameter in the operation.
- It is a connection between classes that only uses another class as a parameter to an operation.
- In the Fig. 5.7 (b), CourseSchedule is dependent on Course class.
- A dependency can have a name, although names are rarely needed unless you have a model with many dependencies and you need to refer to or distinguish among dependencies.

5.3.2 Generalization [S-18]

- A generalization is a relationship between a general kind of thing (called the superclass or parent) and a more specific kind of thing (called the subclass or child).
- Generalization is a relationship between a superclass (parent) and a subclass(child) which is also known as "is-a-kind of" relationship.
- Generalization means objects of the subclass may be used anywhere; the superclass may appear, but not in reverse.
- A class may have zero, one or more parents. A class that has no parents and one or more children is called a root class or base class.
- A class without children (subclasses) is called as a leaf class.
- Generalization relationship basically describes an inheritance relationship where properties of the base class will be inherited by the derived class.
- A class that has only one base class (parent) represents a Single inheritance and a class with more than one base class represents Multiple inheritance.

Representation:

- Generalization is represented as a solid directed line with a large open/hollow arrowhead pointing to parent class.

Example:

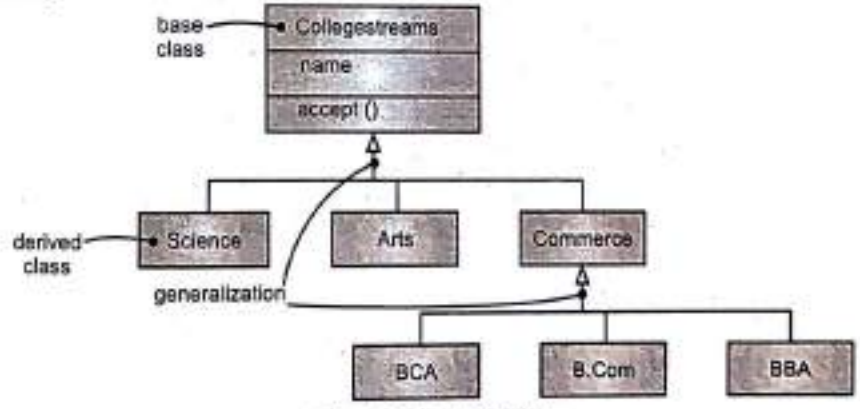


Fig. 5.8: Generalization

- In the Fig. 5.8, Collegestreams is a base class from which Science, Arts and Commerce classes are derived. The derived classes inherit the properties of the base class.
- From the derived class Commerce which plays a role of base class, new classes are derived i.e. BCA, BBA and B.Com which inherit the properties of base class Commerce.

5.3.3 Association [S-18, W-18]

- An association is a structural relationship that specifies that objects of one thing are connected to objects of another.
- Association is basically a set of links that connects elements of an UML model. It also describes how many objects take part in that relationship.
- We can say association relationship connects two classes which means that objects of both classes can communicate with each other. An association may be a binary or n-ary i.e. association is between two classes or may be between more than two classes.
- Generalization relationship is nothing but an inheritance where properties of the base class will be inherited by the derived class.

Representation:

- Graphically, an association is represented as a solid line connecting to same or different classes.
- A name for the association relationship can be specified over the line.
- In addition, a triangle near the relationship name indicates the direction in which the relationship name has to be read.

Example:

- Here is an example (Fig. 5.9) representing the association relationship between the two entities Person and Company.
- The association relationship is given a name Works for and the relationship is read as 'Person works for Company' as specified by the triangle signifying the direction.

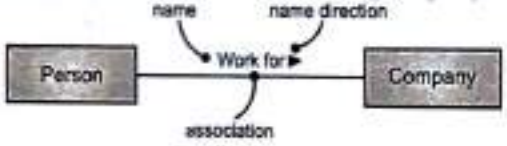


Fig. 5.9: Association relationship

5.3.3.1 Important Terms in Association

- Along with association type of relationship, we can show more information which specifies details of association. There are four such ways called *adornments* as follows.

1. Name:

- Association can have a name which is used to describe the nature of relationship, which will help to clearly understand the relationship.
- A direction triangle is used along with the name which points in the direction we intend to read the name as shown in Fig. 5.10 (a).

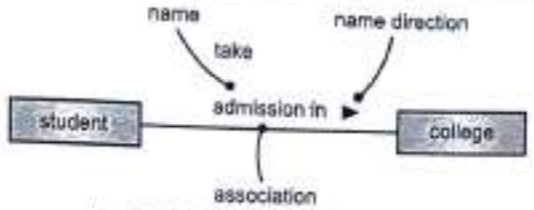


Fig. 5.10 (a): Association with name

2. Role:

- Association relationship is between classes.
- A role name is a name written at one end of an association.
- Fig. 5.10 (b) shows role names for an association. A person assumes the role of employee with respect to a company and a company assumes the role of employer with respect to a person.

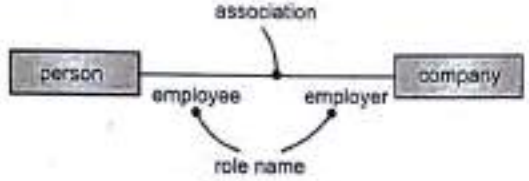


Fig. 5.10 (b): Role

- Remember the following points regarding role names:
  - Role names are necessary for association between two objects of the same class.
  - It is useful to distinguish between two associations of the same air of class.
  - All role names on the far end of associations attached to the class must be unique.
  - It is really a derived attribute of the source class. So it should be unique.
  - No role name should be the same as an attribute name of the source class.
  - Role names do not represent derived attributes of the participating classes.

5.3.3.2 Types of Association [W-18]

- There are two types of Associations: 1. Aggregation, 2. Composition.

1. Aggregation:

- Aggregation refers to the formation of a particular class as a result of one class being aggregated or built as a collection.
- Aggregation is a special form of association. It is a simple concept which distinguishes a "whole" from a "part".
- This type of relationship is also called a "has-a" relationship, which means that an object of whole has objects of parts.

Representation:

- This type of relationship is represented by showing a plain association with an open diamond at the whole end. That means, draw a line from the parent class to the child class with a diamond shape near the parent class.

Example:

- The class college is made up of one or more departments.

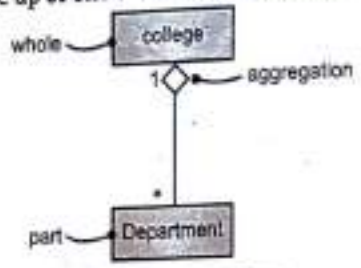


Fig. 5.11: Aggregation

- In the above Fig. 5.11, the whole/part association is represented by aggregation along with multiplicity, one college has many departments is one meaning and departments are part of the whole college is shown with aggregation.

2. Composition:

- It is very similar to the aggregation relationship, with the only difference being its key purpose of emphasizing the dependence of the contained class to the life cycle of the container class.
- That is, the contained class will be destroyed when the container class is destroyed. For example, House (parent) and Room (child). Room doesn't exist separate from a House.

Representation:

- To describe a composition relationship in a UML diagram, use a directional line connecting the two classes, with a filled diamond shape adjacent to the container class and the directional arrow to the contained class.

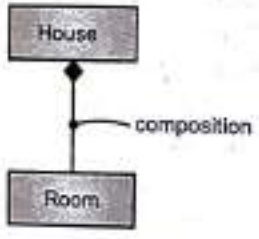


Fig. 5.12: Example of Composition Relationship

5.4 COMMON MECHANISM [S-18, W-18]

- UML is an open-ended language. It is possible to extend the capabilities of UML in a controlled manner to suit the requirements of a system.
- Common mechanism is a way to give more details to understand the UML models. UML made it easy by having four common mechanisms that are regularly applied throughout the language i.e. specifications, adornments, common division and extensibility mechanisms.
- UML's extensibility permits it to extend the language in a controlled way, which includes stereotypes, tagged values and constraints.
- Stereotypes, Tagged values and Constraints are used to add new building blocks, to create new properties and specify new semantics.

1. Note:

- A note is a notational thing in UML.
- Notes are the most important kind of adornment which is used to attach information to a model such as requirements, observations, reviews and explanations.
- Note type of thing is used to show comments for an element or a group of elements.
- This way we use UML to organize all artifacts that are generated or used during development as shown in Fig. 5.13.

Representation:

- Graphically, the Note is shown as a rectangle with a dog-eared corner with textual or graphical comment.

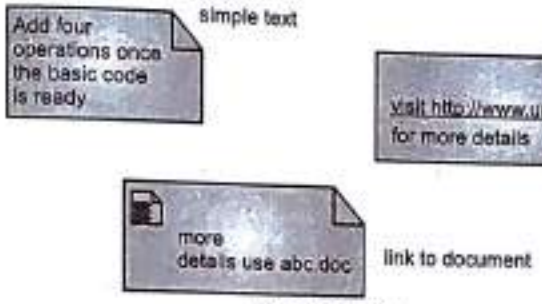


Fig. 5.13: Notes

2. Stereotype:

- It is an extension of basic building blocks which is similar one to the existing one but more specific to our problem.
- When we stereotype an element like class or note, we are actually extending the UML by creating a new building block just like existing one but with special properties such as tagged values, semantics(constraints) and notation (each stereotype may provide its own icon).

Representation:

- A stereotype is represented as a name enclosed by guillemets (e.g. `<< name >>`) and placed above the name of another element.
- To see and get the meaning, an icon can be defined for stereotypes and show that icon to the right of name or use that icon as the basic symbol for stereotyped items as shown in Fig. 5.14.

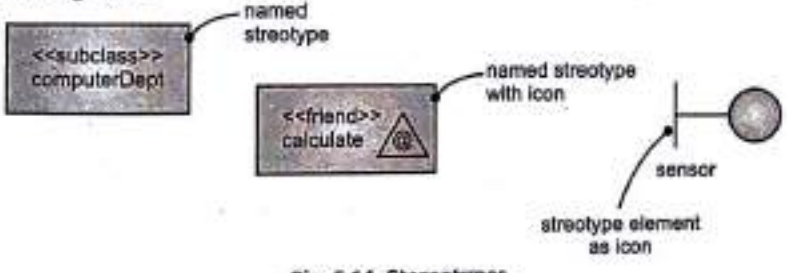


Fig. 5.14: Stereotypes

S-19

3. Tagged Values:

- A tagged value is a property of a stereotype, allowing us to create new information in an element bearing that stereotype.
- We can define tags for existing elements of UML or we can define tags that apply to separate stereotypes that have tagged value.
- Tagged value is a metadata i.e. data about data.

Representation:

- A tagged value is represented as a string enclosed in curly brackets { } and placed below the name of another element.
- The string includes a name (i.e. tag), a separator (i.e. symbol =) and a value as shown in Fig. 5.15.

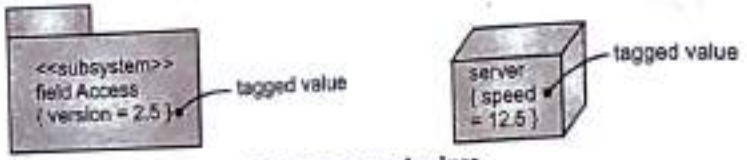


Fig. 5.15: Tagged values

4. Constraints:

- Constraint means rule or condition. With this type of mechanism, we can add new rules or modify existing ones which gives us a clear meaning of the model.
- Constraints represent restrictions placed on an element.

Representation:

- Graphically, constraint is represented as a string enclosed by curly brackets { } and placed near the associated elements or connected to that element or elements by dependency relationships. It can even be represented in a note.
- A constraint specifies conditions that must be well formed as shown in Fig. 5.16. It is shown with relationships.

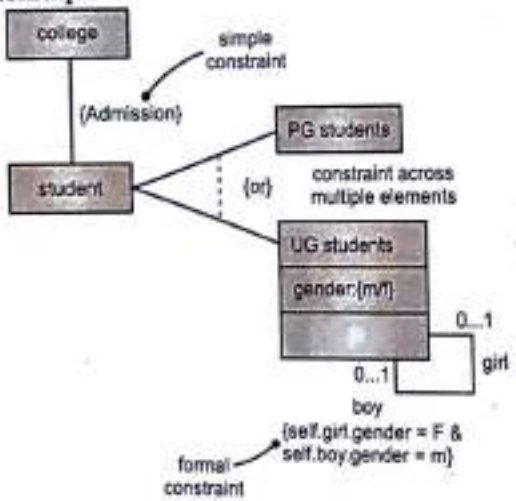


Fig. 5.16: Constraints

5.4.1 Important Concepts of Common Mechanism

1. Class Template:

- UML classes could be templated or bound.
- The example below shows template class Array with two formal template parameters. The first template parameter T is an unconstrained class template parameter. The second template parameter n is an integer expression template parameter.
- Template binding for the bound class Customers substitutes the unconstrained class T with class Customer and boundary n with integer value 24. Thus, the bound class Customers is an Array of 24 objects of Customer class.

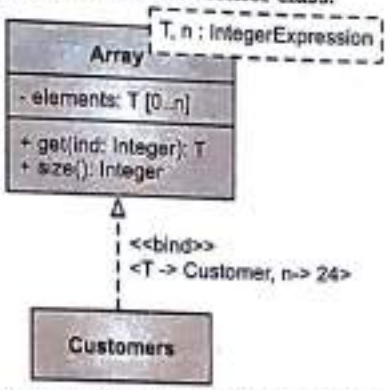


Fig. 5.17: Template class: Array and Bound class: Customers

2. Link Attribute:

- A link attribute is a property of the links in an association.
- It is a property of the objects in a class similar to the attribute.

Representation:

- The OMT (Object Modeling Technique) notation for a link attribute is a box attached to the association by a loop symbol.
- One or more link attributes may appear in the second region of the box.
- This notation shows dissimilarity between attribute for objects and attribute for links.

Fig. 5.18: Link attribute for a Many-to-Many Association

Example:

- Fig. 5.18 shows an example of a link attribute. In this fig. `Access permission` is an attribute accessible by link.
- `Access permission` is a point property of file and user. It can not be attached to either file or user alone without losing information.

3. **Ordering:**

- Ordering is a special kind of facility available in OMD (Object Model Diagram).
- Usually, the object on the 'many' side of (Side) an association has no external order.

Representation:

- It can be considered as a set. Sometimes, the objects are externally ordered. Externally ordered objects are denoted by writing (ordered) on the "many" end of an association i.e. next to the multiplicity dot for the role.

Example:

- Fig. 5.19 shows an example of ordering. The windows are externally ordered, so only the topmost window is visible at any point on the screen.

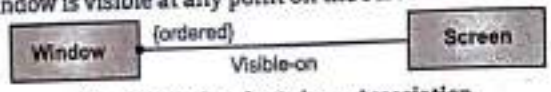


Fig. 5.19: Ordered sets in an Association

4. **Association Classes:**

- Association classes are used to identify information about an association.

Representation:

- The information is placed in a class attached to the association that it describes by a dashed line.
- An association class can be seen as an association that also has class properties or as a class that has association properties.

Example:

- Fig. 5.20 shows an Association class.

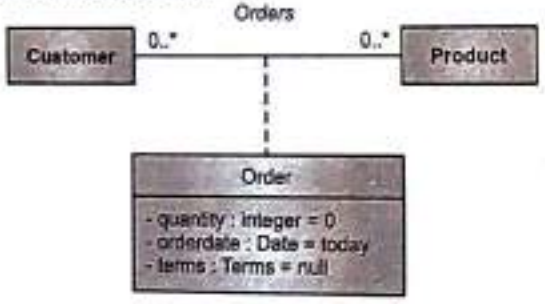


Fig. 5.20: Association class

- Be on the viewpoint for association classes when you see a multiplicity of more than one used on both ends of the association.

5.5 CLASS DIAGRAM [S-19]

- The class diagram is a static diagram. It represents the static view of an application or a system.
- The class diagram describes the attributes and operations of a class. It also describes the constraints imposed on the system.

- The class diagrams are widely used in the modeling of object oriented systems.
- Class diagram provides an overview of the target system by describing the objects and classes inside the system and the relationships between them.
- Class diagrams provide a wide variety of usages; from modeling the domain-specific data structure to detailed design of the target system.

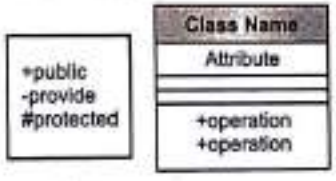
Purpose of the Class Diagram:

- Analysis and design of the static view of an application.
- Describes responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

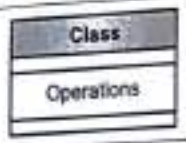
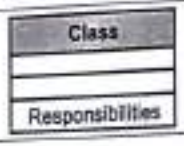
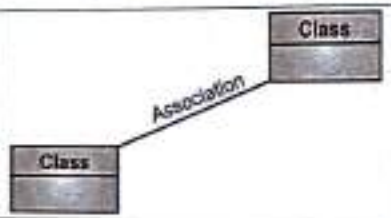
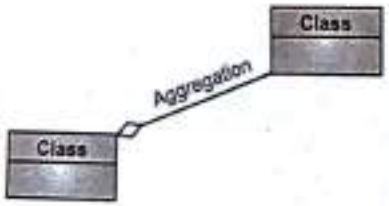
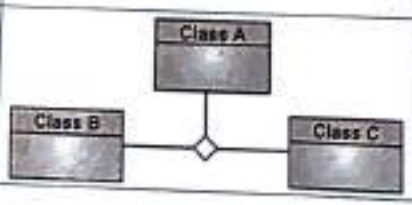
5.5.1 Notations

- Following table gives the basic Notations (symbols) used for Class diagrams.

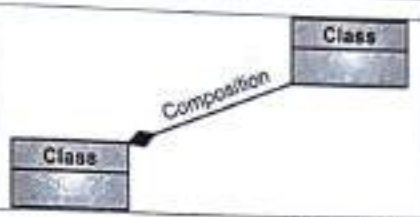
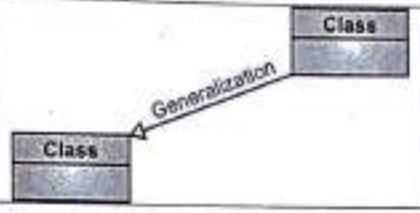
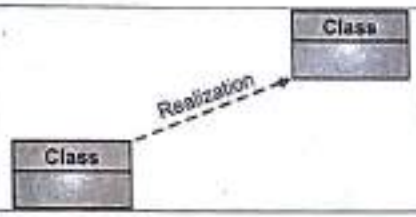
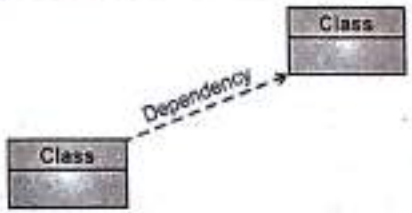
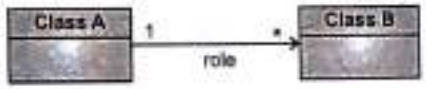
Table 5.1: Notations (symbols) used for Class Diagrams

Name	Symbol	Description
1. **Class**		A class describes a set of objects that share the same specifications of features, constraints, and semantics.
2. **Active class**		Active classes initiate and control the flow of activity, while passive classes store data and serve other classes. Illustrate active classes with a thicker border.
3. **Visibility**		Use visibility markers to signify who can access the information contained within a class. Private visibility hides information from any-thing outside the class partition. Public visibility allows all other classes to view the marked information. Protected visibility allows child classes to access information they inherited from a parent class.

contd. ...

4. Attribute		Attribute is a typed value that defines the properties and behavior of the object.
5. Operation		Operation is a function that can be applied to the objects of a given class.
6. Responsibility		Responsibility is a contract which the class must conform to.
7. Interface		Interface is an abstract class that defines a set of operations that the object of the class associated with this interface provides to other objects.
8. Association		Association is a relationship that connects two classes.
9. Aggregation		Aggregation is an association with the relation between the whole and its parts, the relation when one class is a certain entity that includes the other entities as components.
10. N-ary Association		N-ary association represents two or more aggregations.

contd. ...

11. Composition		Composition is a strong variant of aggregation when parts cannot be separated from the whole.
12. Generalization		Generalization is an association between the more general classifier and the more special classifier.
13. Inheritance		Inheritance is a relationship when a child object or class assumes all properties of its parent object or class.
14. Realization		Realization is a relationship between interfaces and classes or components that realize them.
15. Dependency		Dependency is a relationship when some changes of one element of the model can need the change of another dependent element.
16. Multiplicity		Multiplicity shows the quantity of instances of one class that are linked to one instance of the other class.
17. Package		Package groups the classes and other packages.

contd. ...

18. Collaboration		A Collaboration specifies a view (or projection) of a set of cooperating classifiers. It describes the required links between instances that play the roles of the collaboration, as well as the features required of the classifiers that specify the participating instances.

5.5.2 Examples

[S-18, W-18, S-19]

- Class diagrams are widely used to describe the types of objects in a system and their relationships.
- Class diagrams model class structure and contents using design elements such as classes, packages and objects.
- Class diagrams describe three different perspectives when designing a system, Conceptual, Specification and Implementation. These perspectives become evident as the diagram is created and help solidify the design.
- Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So a collection of class diagrams represents the whole system.

1. Class Diagram for Railway Reservation System:

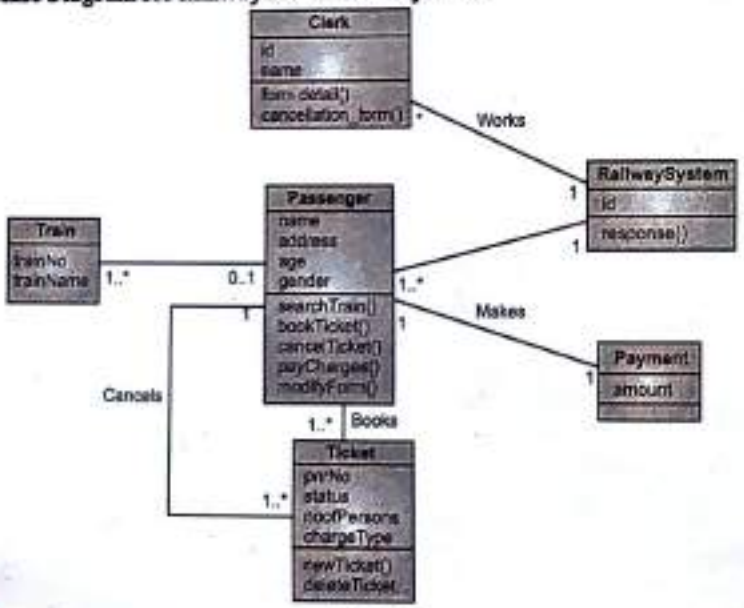


Fig. 5.21

2. Class Diagram for Library Management System:

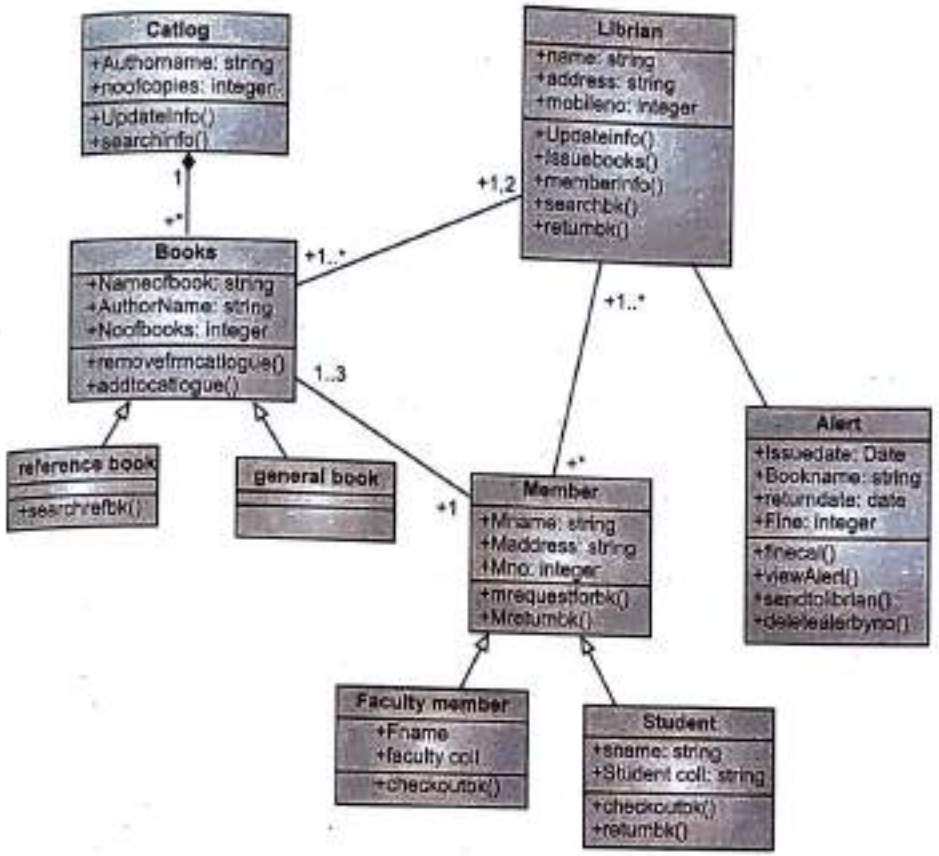


Fig. 5.22

3. Class Diagram for ATM:

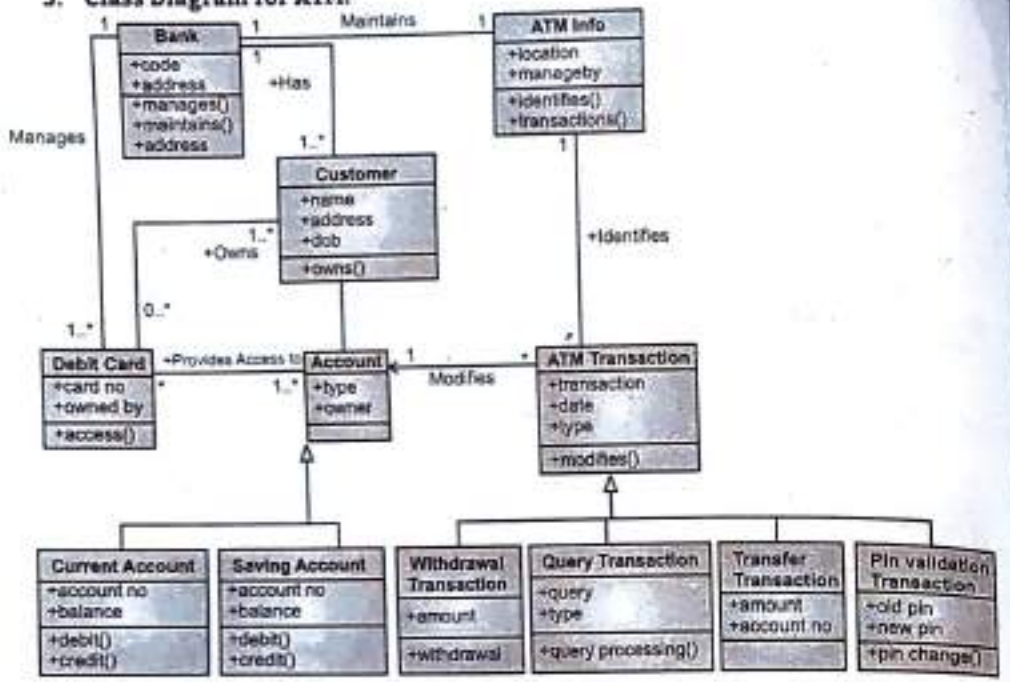


Fig. 5.23

4. Class Diagram for a Simple Bank System:

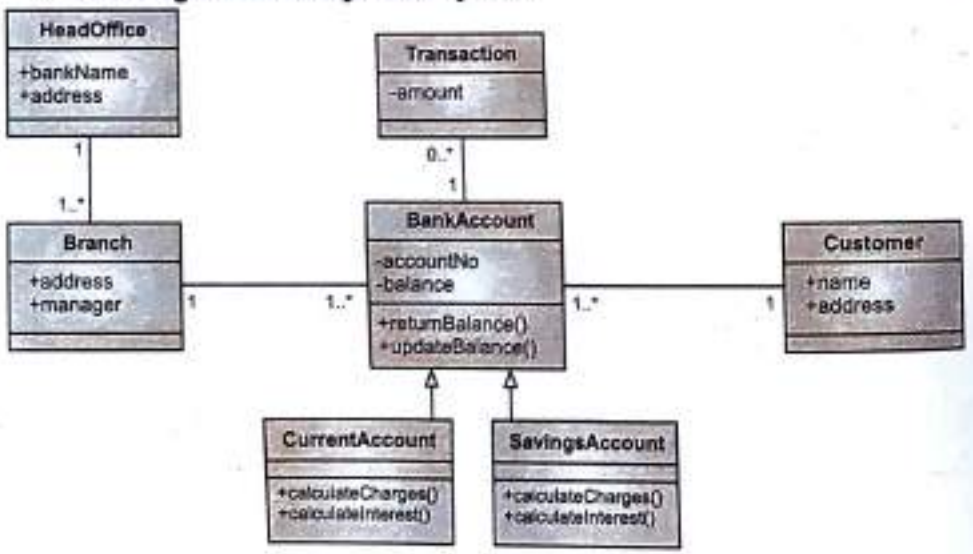


Fig. 5.24

5. Class Diagram for Hospital Management System:

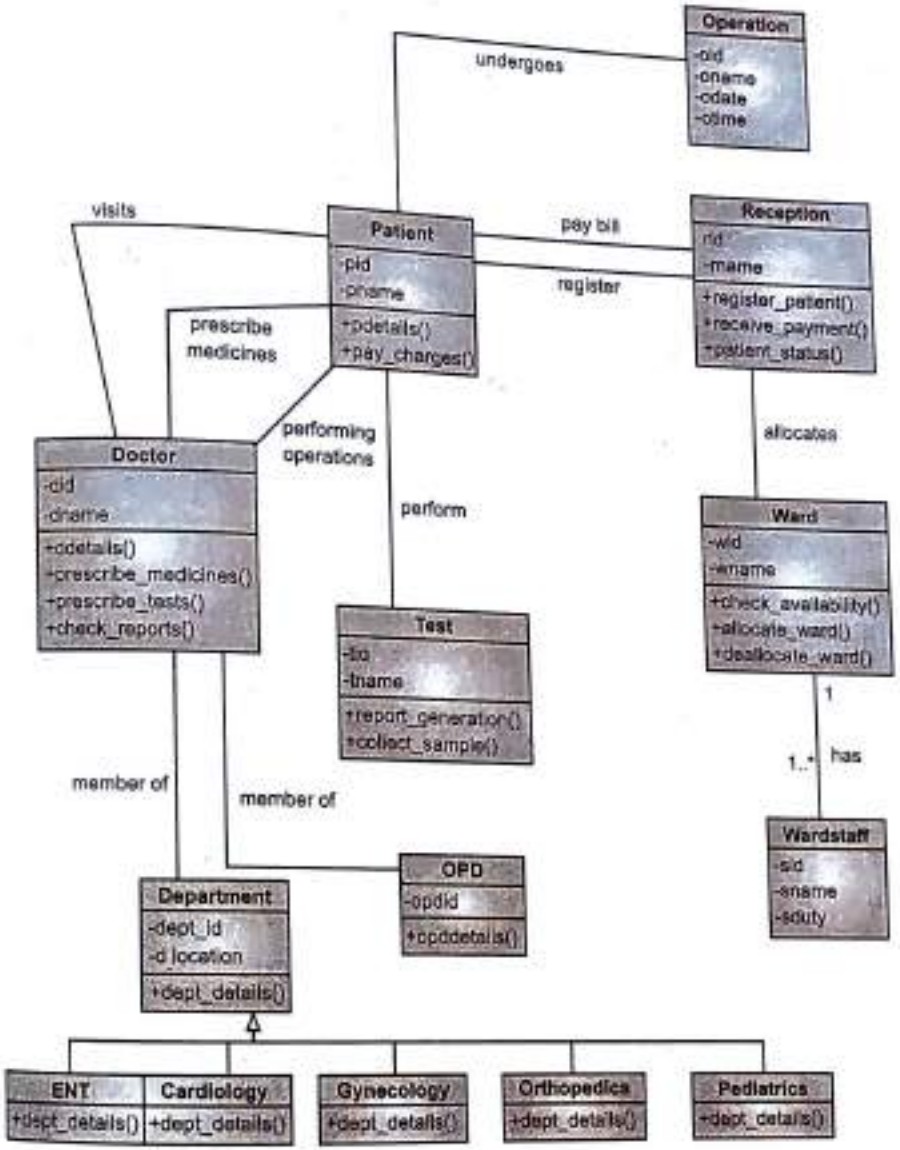


Fig. 5.25

5.6 INTRODUCTION TO ADVANCED STRUCTURAL MODELING

- The discipline of modeling in OOSE has advanced only slowly compared to disciplines concerned with analyzing and solving models once they are brought into being. Structured Modeling is an attempt to restore this imbalance.
- Structured modeling in OOSE aims to provide a formal mathematical framework and computer-based environment for conceiving, representing and manipulating a wide variety of models.
  1. **Structured modeling framework** uses a hierarchically organized, partitioned and attributed acyclic graph to represent the semantic as well as mathematical structure of a model.
  2. **The Computer-based Environment** is evolving via experimental prototypes that provide for ad hoc query, immediate expression evaluation, solving simultaneous systems and optimization.
- The UML provides some advanced properties to be attached to classes to represent a class in more detail. We can call these classes as advanced classes as it permits us to visualize, specify, construct and document a class to any level of detail we wish.
- A view of a system that highlights the structure of the advanced objects includes their classifiers, advanced relationships, attributes and operations.
- In this chapter we study various concepts of Advanced Structural Modeling.

5.7 ADVANCED CLASSES

- A class represents a graph of things that have common (same) state and behaviour.
- The UML provides a representation for a number of advanced properties as shown in Fig. 5.26.
- The notation in Fig. 5.26 permits us to visualize, specify, construct and document a class to any level of detail we wish, even sufficient to support forward and reverse engineering of models and code.

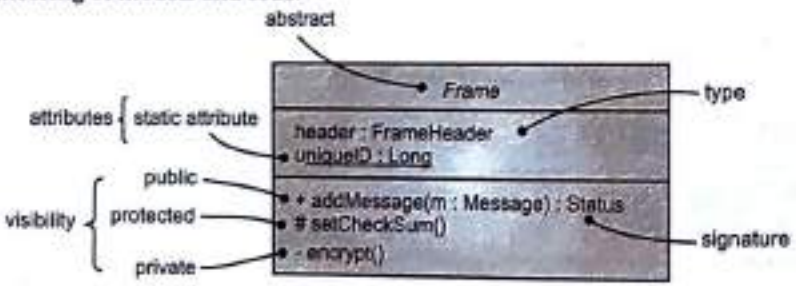


Fig. 5.26: Advanced Classes

- We specify a class is abstract by writing its name in italics (See Fig. 5.26).
- In contrast, a concrete class is one that may have direct instances. Visibility describes whether an attribute or operation is visible and can be referenced from classes other than the one in which they are defined.

- Visibility is of three types i.e., Public, Private and Protected.
- To provide greater control over our model and detail in our abstraction, the attributes of a class can be specified according to scope, multiplicity, visibility, type, initial value, and changeability for each.

5.7.1 Classifier [S-19]

- A classifier is a mechanism that has structural features (in the form of attributes), as well as behavioral features (in the form of operations).
- Classifiers include class, data types, signals, components, nodes, use cases and subsystems.
- The modeling elements that can have instances are called classifiers.
- A classifier has structural features i.e. in the form of attributes along with behavioral features i.e. in the form of operations.
- Different classifiers are shown and explained as shown in Fig. 5.27.
  1. **Class:** It is a description of objects that share the same attributes, operations, relationships and semantics.
  2. **Interface:** A collection of operations that are used to specify a service of a class or a component.
  3. **Data type:** A type whose values have no identity, which includes basic built-in types as well as enumeration types (i.e. Boolean).
  4. **Signal:** The specification of an asynchronous stimulus communicated between instances.

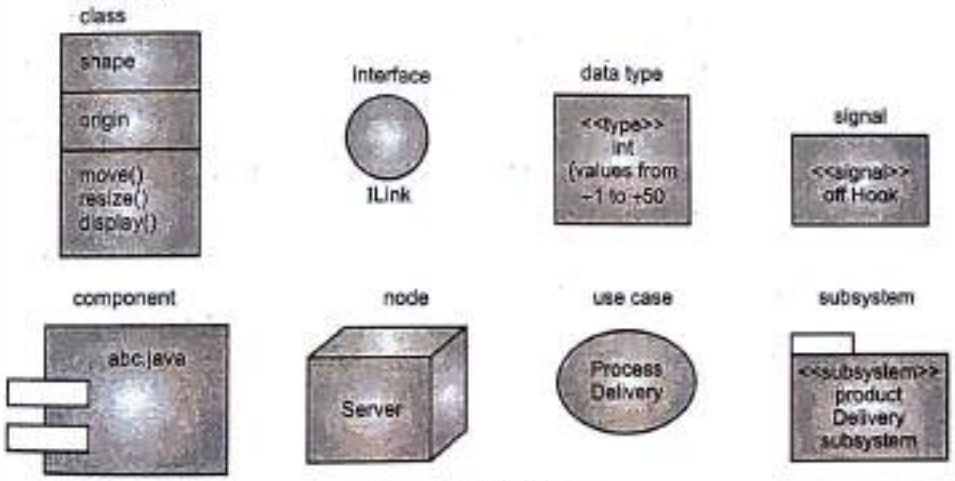


Fig. 5.27: Classifiers

  5. **Component:** A physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces.
  6. **Node:** A physical element that exists at run time and that represents a computational resource, having at least some memory and processing capability.

7. **Use case:** It is a description of a sequence of actions including variants that a system performs which produces observable results of value to a particular actor.
8. **Subsystem:** A grouping of elements of which some constitute a specification of behaviour offered by other contained elements.

5.7.2 Attributes and Operations

1. **Attributes:**

- A class's structural features are indicated by its attributes.
- At the abstract level, a class is represented by a writing attribute name which is more than sufficient for the general reader to understand the model.
- Along with these attributes, we can specify visibility, scope and multiplicity of each attribute. Along with these, we can even specify the type, initial value and changeability of each attribute.
- Three defined properties that can be used with attribute values are given in following table:

Table 5.2: Properties of Attributes

Property	Meaning
1. changeable	There are not restrictions on modifying the attribute value.
2. addOnly	For attributes with a multiplicity greater than one, additional values may be added, but once created; a value may be removed or altered.
3. frozen	The attribute value may not be changed after the object is initialized.

2. **Operations:**

- At the abstract level, the operation name plays a role of behavioral feature which is enough at the initial level for the general reader to understand.
- We can even specify the visibility and scope of each operation. We can also specify parameters, return type, concurrency, semantics and other properties of each operation which is collectively called 'operation signature'.
- The **syntax** of operation in UML is:

```
[visibility] name [(Parameter-list)] [: return type] [{Property-string}]
```

Examples:

display	Name
+ display	Visibility and Name
set (n: Name, s: String)	Name and Parameters
getID ( ): Integer	Name and Return Type

5.7.3 Visibility

[S-18]

- Classifier's attributes and operations details can be specified with its visibility.
- Visibility indicates whether the attributes and operations of a classifier can be used by any other classifiers.

- There are three levels of visibility in UML used to specify the details of attributes and operations i.e., public, protected and private.
  - **Public:** A public member is visible from anywhere in the system. In the class diagram, it is prefixed by the symbol '+'.
  - **Private:** A private member is visible only from within the class. It cannot be accessed from outside the class. A private member is prefixed by the symbol '–'.
  - **Protected:** A protected member is visible from within the class and from the subclasses inherited from this class, but not from outside. It is prefixed by the symbol '#'.
- **Example:** Let us consider the `Circle` class in Fig. 5.28. The attributes of Circle are `x-coord`, `y-coord`, and `radius`. The operations are `findArea()` and `findCircumference()`. Let us assume that x-coord and y-coord are private data members, radius is a protected data member, and the member functions are public.

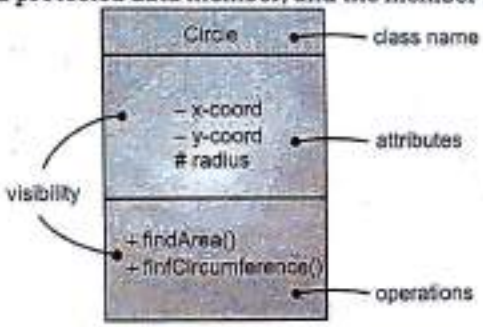


Fig. 5.28: Visibility

- Visibility mode helps to hide implementation details of a classifier. When no visibility is specified, it is by default public visibility.

5.7.4 Scope

- Other important details you can specify for a classifier's attributes and operations are its owner scope.
- The owner scope of a feature (attribute/operations) specifies whether the feature appears in each instance of the classifier or whether there is just a single instance of the feature for all instances of the classifier.
- Owner scope is of two types as explained below and shown in Fig. 5.29.
  1. **Instance:** Each instance of the classifier holds its own value of the feature.
  2. **Classifier:** There is no value of the feature for all instances of the classifier.

Fig. 5.29: Scope

- The class scope is represented by underlining the features name. No adornment means that the feature is instance-scoped.

5.7.5 Abstract, Root, Leaf and Polymorphic Elements

- The properties of base class can be derived by a derived class which is shown with a generalization relationship.
- At the top, an abstraction class is shown and specific ones are shown at the bottom level.

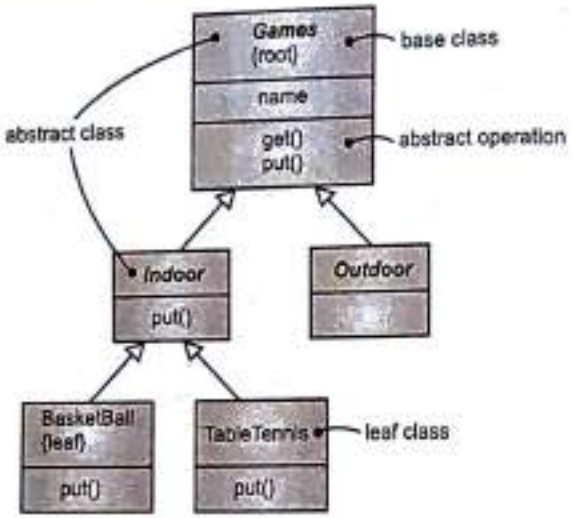


Fig. 5.30: Abstract and Concrete Classes and Operations

- The base class is an abstract class which is represented by writing the name of the class in an italics mode i.e. *Games, Indoor, Outdoor*. Abstract class is one who does not have any direct instances.
- A concrete class is a one (`BasketBall and Table Tennis`) that may have direct instances.
- A leaf class does not have any further children, which can be specified in UML by writing the property leaf below the class name. `BasketBall` is a leaf which has no children.
- Root class is one who does not have any parents and is specified in UML by writing the property root below the class name, i.e. `Games`.
- Operations have similar properties. An operation is polymorphic which means in a hierarchy of classes, you can specify operations with the same signature (declaration) at different points in a hierarchy. For example, in Fig. 5.30, `put()` is a polymorphic operation.

5.7.6 Multiplicity

- Multiplicity specifies how many instances of one class can be associated with how many instances of other class.

- Multiplicity applies to attributes as well. A class having a single instance is called a singleton class. It is indicated in Fig. 5.31.

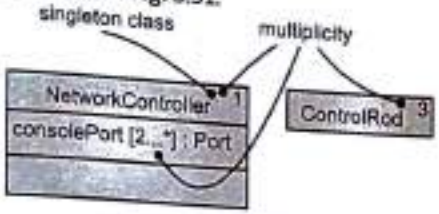


Fig. 5.31: Multiplicity

- Multiplicity is a specification of the range of allowable cardinalities an entity may assume.
- It is specified as a comma-separated list of intervals, where each interval is of the form:

```
minimum..maximum
```

where, `minimum` and `maximum` may be integers or any expression that yields an integer result.

- Some examples of Multiplicity syntax are given in the following table:

Table 5.3: Examples of Multiplicity syntax

Adornment	Semantics
0..1	Zero or 1
1	Exactly 1
0..*	Zero or more
*	Zero or more
1..*	1 or more
1..6	1 to 6
1..3, 7..10,15,19..*	1 to 3 or 7 to 10 or 15 exactly or 19 to many

5.8 ADVANCED RELATIONSHIP

- Relationships are semantic (meaningful) connections between modeling elements.
- To show the relationship between the things, we are using relationship types i.e. dependency, association, generalization and realization.
- The relationship can be complex, to represent the complexity, we need more advanced features.
- Fig. 5.32 shows advanced relationships in UML.

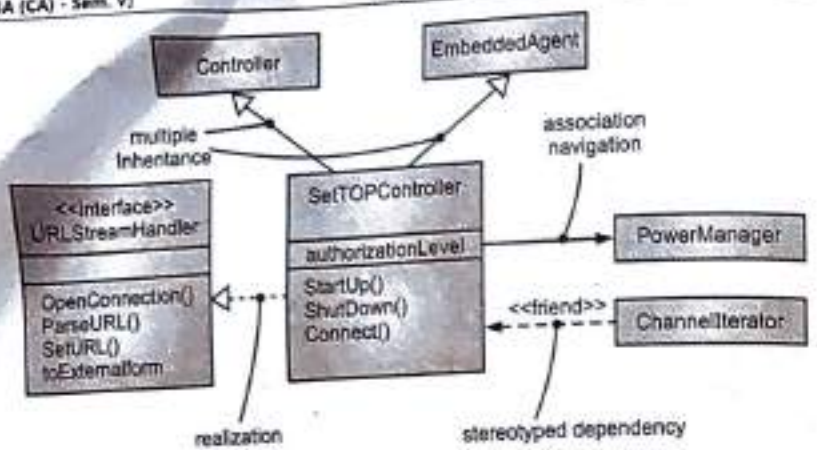


Fig. 5.32: Advanced Relationship

5.8.1 Dependency [S-18]

- A dependency is a relationship between two elements where a change to one element (the supplier) may affect or supply information needed by the other element (the client).
- A dependency is a using relationship, specifying that a change in specification of one thing (e.g. Class `SetTopController`) may affect another thing that uses it (e.g. Class `ChannelIterator`) but not in reverse.
- Graphically, dependency is shown as a dashed line directed to the thing that is depended on.

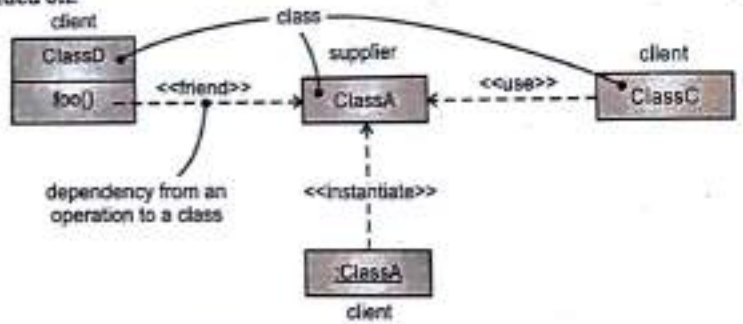


Fig. 5.33: Dependency in Advanced relationship

- To show advanced relationships, UML has defined a number of stereotypes which are organised in six groups.
- Following are the details of stereotypes that can be used to show relationships among different things:

1. **To show relationships among class and objects in the class diagram,** use following stereotypes:
   - **bind:** Specifies that source instantiates the target template using given actual parameters.

   - **derive:** Specifies that the source may be computed from target. When we want to model the relationship between two attributes or two associations.
   - **friend:** Specifies that the source is given special visibility into the target. Use friend when you want to model relationships such as in C++ friend classes.
   - **instanceOf:** Specifies that source object is an instance of the target classifier.
   - **Instantiate:** Specifies that source creates the instance of target.
   - **Powertype:** Specifies that the target is a powertype of source. A Powertype is a classifier whose objects are all children of a given parent. Powertype is used to model classes that cover other classes during modeling databases.
   - **refine:** Specifies that the source is at a finer degree of abstraction than the target. It is used when we want to model classes at different levels of abstraction.
   - **Use:** Meaning of source element depends on the meaning of the public part of the target. Use is applied when we want to mark a dependency as a using relationship.
2. **To show relationship among packages:** Two stereotypes are used: access and import.
   - **access:** Source package is allowed to reference the elements of the target package.
   - **import:** It is a kind of access that specifies the public contents of the target.
3. **Two stereotypes can apply to dependency relationships among use cases:**
   - **extend:** Specifies the target use case extends the behaviour of source use case. i.e. it extends the base(source) use case and adds more functionality to the system.

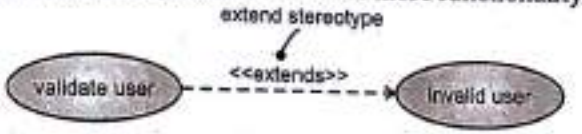


Fig. 5.34: Extend relationship

   - **include:** Specifies that source use case explicitly incorporates the behaviour of another use case at a location specified by source.

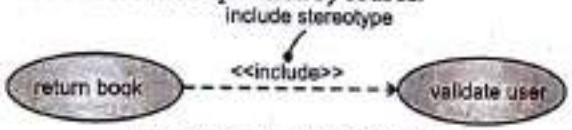


Fig. 5.35: Include relationship

4. **There are three stereotypes which show interaction among objects:**
   - **become:** Specifies that the target is the same object as the source but after some time with possibly different values, state or roles.
   - **call:** Specifies that the source operation invokes the target operation.
   - **copy:** Specifies that the target object is an exact but independent copy of source.
5. **For the State machine, one stereotype is used:**
   (I) **send:** Specifies that source operation sends the target event.
6. **To organize the elements of a system into subsystems, one stereotype is used:**
   (I) **Trace:** Specifies that the target is an historical ancestor of the source.

5.8.2 Generalization

- A generalization is a relationship between a general thing and a more specific kind of that thing. Here, a child class will inherit all the structure and behaviour of the parent and can even add new structure and behavior, or it may modify the behavior of the parent (i.e., overriding).
- Fig. 5.36 shows an example of generalization. This provides the ability to define common properties for derived classes in a common place and use of them depends on the derived class requirements.

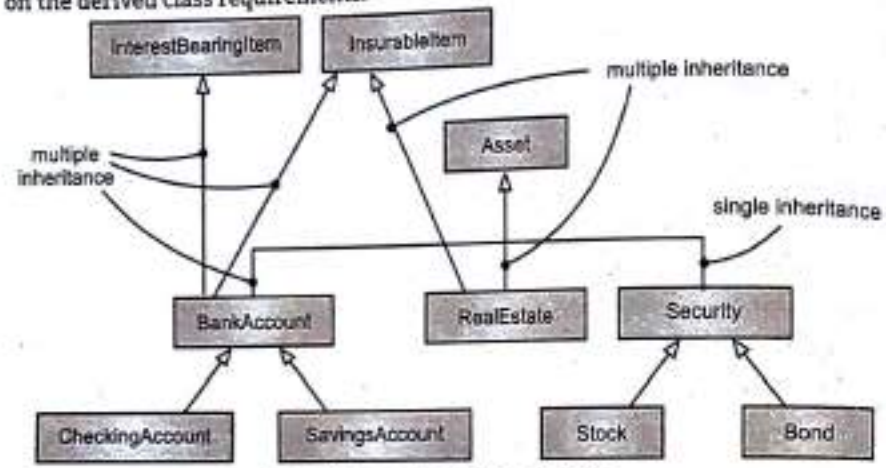


Fig. 5.36: Example of Generalization

- In generalization relationships, instances i.e. objects of the child class may be used anywhere i.e. where instances of parent apply.
- With the generalization relationship, we can show most of the inheritance relationships. Along with this, we can specify more details by using one stereotype and four constraints that may be applied to generalization relationships.

Stereotype:

1. **implementation:** Specifies that the child inherits the implementation of the parent but does not make it public nor support its interfaces. We can use implementation when we want to model private inheritance.

Constraints:

1. **complete:** Specifies that all children in the generalization have been specified in the model and cannot add more children.
2. **incomplete:** Specifies that not all children in the generalization have been specified (even if some are omitted) and that additional children are permitted.
3. **disjoint:** Specifies that objects of parent may have not more than one of the children as a type.

4. **overlapping:** Specifies that objects of parent may have more than one of children as a type.

- Constraint 3 and 4 apply only in case of Multiple inheritance.

5.8.3 Association

- Association is defined as "a structural relationship that conceptually means that the two components are linked to each other".
- Graphically, an association is represented as a solid line connecting same or different classes.

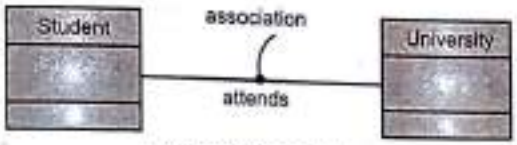


Fig. 5.37: Association

Advanced Properties:

- Other than basic adornments applicable to association i.e. name, role, multiplicity and aggregation, we can add more advanced properties such as navigation, qualification and various types of aggregation.

1. **Navigation:**

- Navigation means traversing/surfing. It is used to mention the direction of association.
- Generally, the association between two classes is bidirectional.
- Sometimes, we have to limit navigation to just one direction. For example, when modeling services of an operating system, you may find an association between User and Password objects.
- Given a user, you want to be able to find the corresponding password objects. While given a password, you want to be able to identify the corresponding user.
- The direction of navigation is shown by the arrow head pointing to the direction of traversal, (See Fig. 5.38).

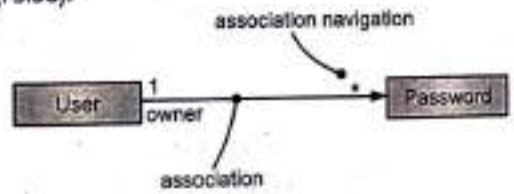


Fig. 5.38: Association Navigation

2. **Multiplicity:**

- Multiplicity limits the number of objects of a class that can be involved in a particular relationship at any point in time. It is a constraint.

3. **Visibility:**
- Visibility means to access permissions given or restricted.
- For an association between two classes, objects of one class can see and navigate to objects of the other, unless otherwise restricted by an explicit statement of navigation.
- For example, association between user group and user and another between user and password (See Fig. 5.40).
- There are visibility modes that are public, private and protected.
- To show the visibility modes for an association, just add a visibility symbol to a role name.
  - **Public visibility:** If any of the visibility symbols is not specified, it is public by default.
  - **Private visibility** indicates that objects at that end are not accessible to any objects outside the association.
  - **Protected visibility** indicates that objects at that end are not accessible to any objects outside the association except for children of the other end as shown in Fig. 5.39.

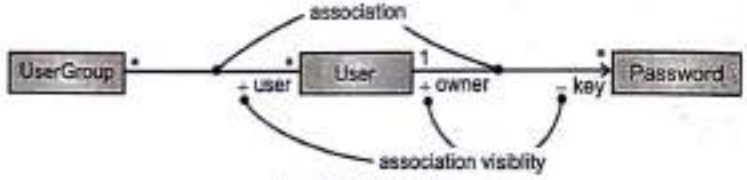


Fig. 5.39: Visibility

4. **Interface Specifier:**
- An interface is a collection of operations that are used to specify a service of a class or a component. Every class may have many interfaces.
- In the context of an association with another target class, a source class may choose to present only part of its face (interfaces) to the world and this part that is chosen is called as the interface specifier.

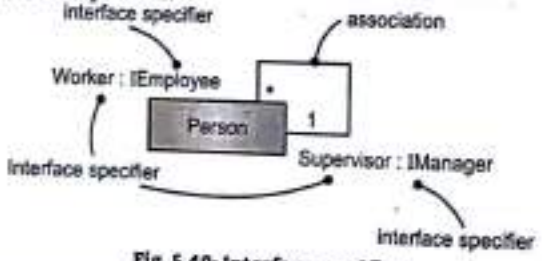


Fig. 5.40: Interface specifier

- In the vocabulary of a HR system (See Fig. 5.40) a `Person` class may implement many interfaces such as `IManager`, `IEmployee` and so on. A relationship between a `supervisor` and `worker` with a One-to-Many association. In the context of this association, a Person in the role of supervisor presents only the `IManager` interface to the worker; a Person in the role of worker presents only the `IEmployee` interface to the supervisor.

5. **Aggregation:**
- It is a simple concept which distinguishes a "whole" from a "part".
- In aggregation there is no life cycle dependency which means both objects can have their separate life. For example, an employee has a home address. But vice versa, the home address has employees, does not make sense.

6. **Composition:**
- Composition is a form of aggregation with strong ownership and coincident lifetime as part of the whole.
- In composite aggregation, an object may be a part of only one composite at a time. For example, in a windowing system, a frame belongs to exactly one window, which is exactly opposite to simple aggregation in which a 'part' may be shared by several 'wholes'.
- In composite aggregation, the whole is responsible for the disposition of its parts, which means that the composite must manage the creation and destruction of its parts i.e. when you create a frame in a windowing system, you must attach it to an enclosing window.
- Similarly, when you destroy the window, the window object must in turn destroy its frame parts as shown in Fig. 5.41.

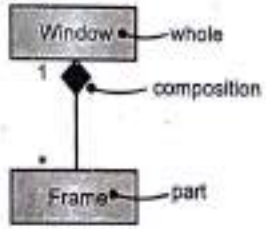


Fig. 5.41: Composition

5.8.4 Realization [S-19]

- A realization is a semantic relationship between classifiers in which one classifier specifies a contract that another classifier guarantees to carry out.
- It is represented as a dashed line with a large open arrowhead pointing to the classifier that specifies the contract, (See Fig. 5.42).
- We may use realization in two situations i.e. in the context of interfaces and in the context of collaborations.
- Realization type of relationship is used to specify the relationship between an interface and the class that provides an operation or service for it.

Representation:
- In system implementation view, realization can be represented in two ways i.e. in canonical form (using interface stereotype and dashed directed line with large open arrowhead) and in an elided form (using interface lollipop notation) as shown in Fig. 5.42.

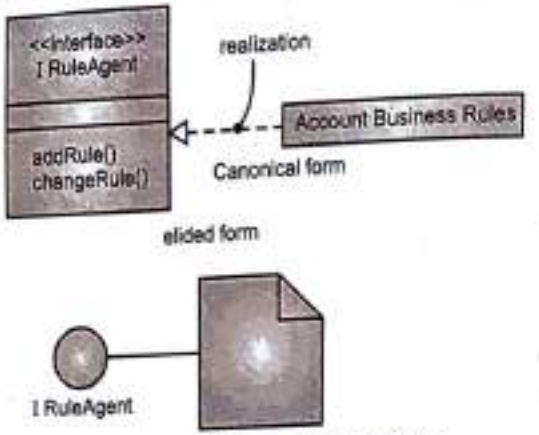


Fig. 5.42: Realization of an interface

5.8.5 Association Classes

- One concern in OO modeling is that when there is a many-to many relationship, sometimes there are certain attributes that can't easily be accommodated in either of the classes.
- Consider the Fig. 5.43(a). What happens if you add the business rule that each Person has a salary with each Company they are employed by? Where the salary should be recorded? In the Person class or in the Company class? In order to deal with this type of situation we come up with an Association class.

Fig. 5.43 (a): Company and Person Class

- An association class is an association that is also a class. It can have attributes, operations, and other associations.
- Instances of the association class are really links that have attributes and operations.
- There can only be one link between any two objects at any point in time in an association class, (See Fig. 5.43 (b)).

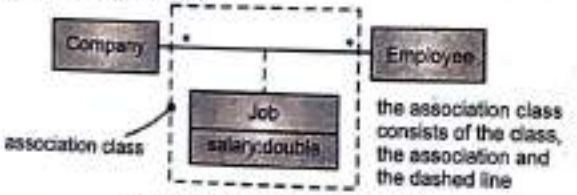


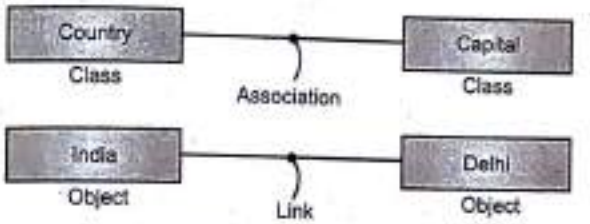
Fig. 5.43 (b): Association classes

- Consider the situation when a given Person object can have more than one Job with a given Company object. Then we have to use a reified association. Reified association allows more than one link between any two objects at a particular point in time.

Fig. 5.43 (c): Reified association

5.8.6 Link and Association

- An instance of association is called as a link.
- A link is a semantic connection between two objects that allows messages to be sent from one object to the other.
- An arrowhead is put on the end of a link to indicate navigability.
- Link is an instance of an association, which means if we connect `Country` class and `Capital` class with association line then we can use link to connect specific County (India) which is an instance of Country class and a capital (Delhi) which is an instance of Capital class.



Class Association Class

Fig. 5.44: Link and Association

5.9 INTERFACE [W-18]

- An interface is a named collection of operations used to specify a service of a class or of a component.
- An interface can be defined as "collection of operation signature and/or attribute definitions that ideally define a cohesive set of behaviors".
- Graphically, an interface is represented by a circle and in its expanded form it may be represented as a stereotyped class, (See Fig. 5.45).
- The notion permits you to visualize the specification of an abstraction apart from any implementation.

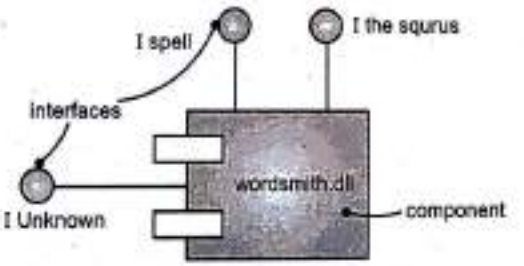


Fig. 5.45: Interfaces

- An interface is graphically represented as a circle in its expanded form. It may be used to specify a contract for a use case or subsystem.

5.9.1 Names

- Every interface must have a name that distinguishes it from other interfaces. A name is string of text consisting of any number of letters, numbers and certain punctuation marks.
- An interface name must be unique within its enclosing package.
- Two naming mechanisms of an interface, a simple name (only name of the interface), a path name is the interface name prefixed by the name of the package in which that interface lives, represented in Fig. 5.46.
- To distinguish an interface from a class, prepend an 'I' to every interface name.

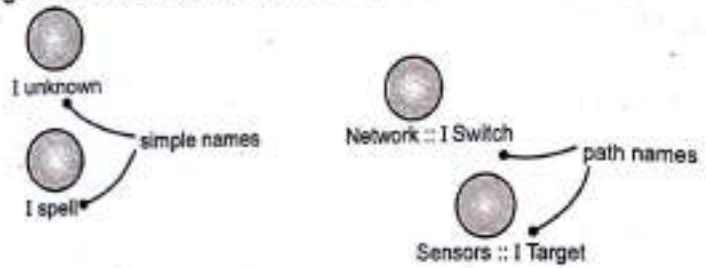


Fig. 5.46: Simple and Path Names

- Fig. 5.47 shows naming an interface.

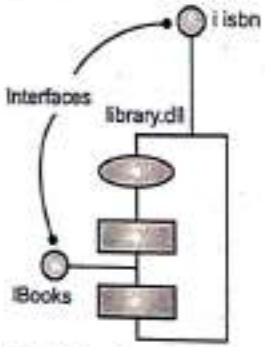


Fig. 5.47: Naming an Interface

5.9.2 Operations

- Interface does not have any structure like class. An interface may have any number of operations.
- These operations may be shown with visibility properties, stereotypes, concurrence properties, tagged values and constraints.
- An interface represented by a circle hides the display of operations.
- For better understanding, an interface can be represented as a stereotyped class by showing its operations in the appropriate section as shown in Fig. 5.48.

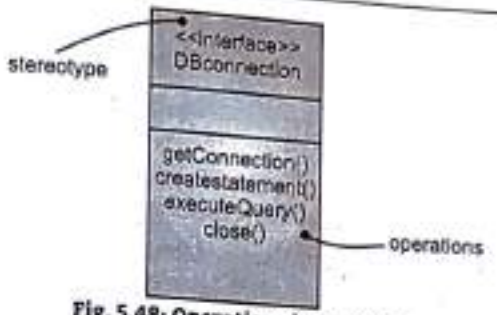


Fig. 5.48: Operations in Interface

5.9.3 Relationships

- The association between two classes is called a relationship.
- An interface may participate in different types of relationships like association, dependency, generalization and realization as shown in Fig. 5.49.
- Association is a simple relationship between two classes. Aggregation is a special type of Association. It is known as the "Has-A" relationship. Composition is a special type of Aggregation. It is known as an "Is-A" relationship.
- Generalization is a relationship between a Parent and its Derived class. It is nothing but inheritance.
- Realization is a relationship between the blueprint class (Interface) and a class containing its respective implementation details. Change in structure of a class affects other classes then there is a dependency between those classes.

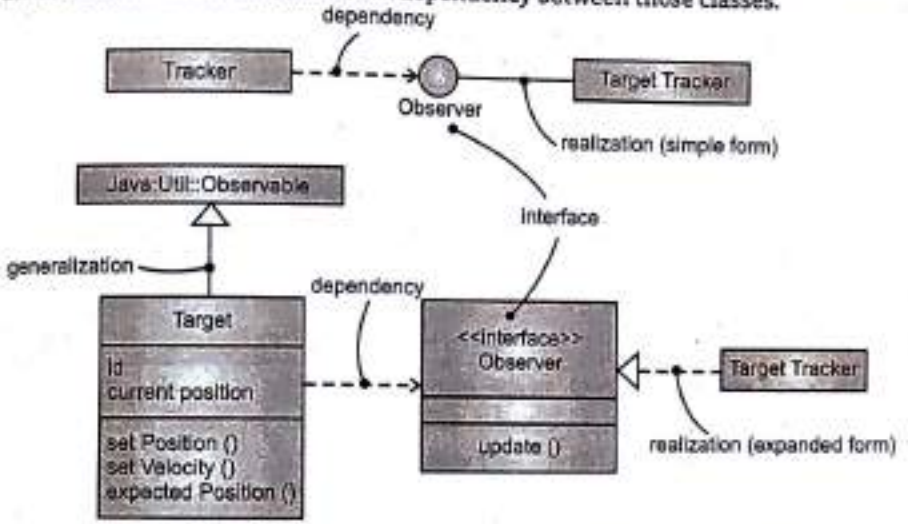


Fig. 5.49: Relationships

5.10 TYPES AND ROLES

1. **Type:**

- A type is a stereotype of a class used to specify a domain of objects, together with the operations applicable to the object of that type.

- To distinguish a type from an interface (or a class) prepend a 'T' to every type.
- Stereotype type is used to formally model the semantics of an abstraction and its conformance to a specific interface.

2. **Role:**

- A role name (indicates) a behavior of an entity participating in a particular context. Or, a role is the face that an abstraction presents to the world.
- For example, consider an instance of the class Person. Depending on the context, that Person instance may play the role of Mother, Employee, Manager, Customer and so on.
- When an object plays a particular role, it presents a face to the world, and clients that interact with it expect a certain behavior depending on the role that it plays at the time.
- For example, an instance of Person in the role of Manager would present a different set of properties than if the instance were playing the role of Mother.
- Fig. 5.50 indicates the role an Employee played by a Person and is represented statically there.

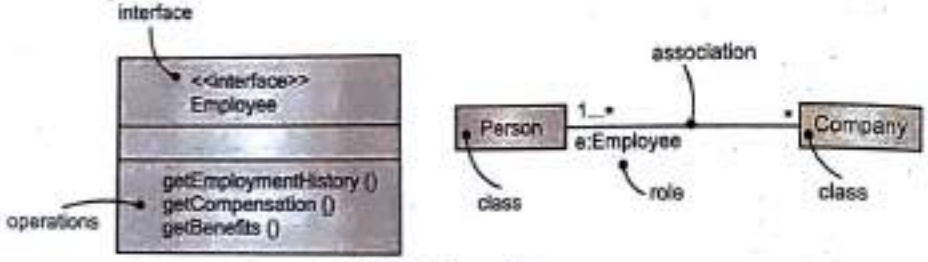


Fig. 5.50: Roles and Types

- In Fig. 5.51 the Person presents the role of Employee to the Company, and in that context, only the properties specified by Employee are visible and relevant to the Company.

5.11 PACKAGES [S-18, 19]

- A package is a general purpose mechanism for organizing modeling elements into groups: Use case, Actors, classes, Components.
- A package may contain other packages, thus providing for a hierarchical organization of packages.
- Graphically, a package is shown as a tabbed folder as shown in Fig. 5.51.

Fig. 5.51: Notation of Package

5.11.1 Names

- Every package must have a name that distinguishes it from other packages.
- Package name is a textual string, consisting of any number of letters, numbers and certain punctuation marks except colon.
- A package may be represented by two ways: simple name and path name. (See Fig. 5.52).
- A simple name is a package name alone while a path name is the package name prefixed by the name of the package in which that package lives.

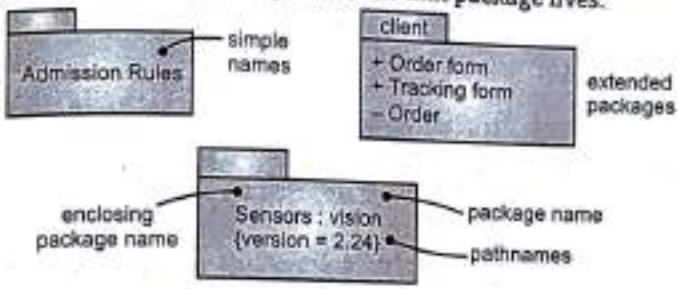


Fig. 5.52: Simple and Extended Packages

5.11.2 Owned Elements

- A package may own different elements like classes, interfaces, components, nodes, collaborations, use case diagrams and even other packages.
- Owning is a composite relationship which means that the element is declared in the package. If a package is destroyed, the element is destroyed. Every element is uniquely owned by exactly one package.
- A package that forms a namespace of the same kind must be named uniquely within the context of its enclosing package.
- Packages may have other packages. This means it is possible to decompose models hierarchically.

Fig. 5.53: Owned Elements

5.11.3 Visibility

- The visibility of attributes and operations owned by class can be controlled in the same way the visibility of packages can be controlled.

- An element owned by a package is public, which means that it is visible to the contents of any package that imports the elements enclosing the package.
- Protected elements can be seen by children (i.e. the derived class only) while private elements cannot be seen outside the package in which they are declared.
- The visibility of an element owned by package can be specified by prefixing the element's name with an appropriate visibility symbol.
- Public elements are shown by prefixing the name with the + symbol.
- Protected and Private elements are shown by prefixing elements name with # symbol and - symbol respectively.
- Visibility controls the accessibility of individual elements of the package in case of import or export.

Table 5.4: Types of visibility of Packages

Visibility	Public	Protected	Private
packages	Any package can access it.	Can be used by only children.	Cannot be accessed by outside or the package is sealed.

- **For example:**

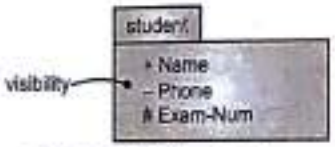


Fig. 5.54: Visibility of Package

5.11.4 Importing and Exporting

Importing:

- Importing grants one-way permission for the elements in one package to access the elements in another package.
- In UML, an import relationship as a dependency is shown with the stereotype import.
- Importing means accessing the elements of the source by the target. It can be done by using the stereotype `<<import>>`. It's a one way process. It's non- transitive. If X's package imports Y's package, X can now see Y, although Y cannot see X.
- A package import is shown using a dashed arrow with an open arrowhead from the importing namespace to the imported package.

Exporting:

- The public parts of a package are called its Exports. The parts that one package exports are visible only to the contents of those packages that import the package as shown in Fig. 5.55.

- In the Fig 5.56, Package `GUI` exports one class, i.e. `Window`. Event handler is not exported by `GUI`. `Event Handler` is a protected part of the package. Package `Policies` explicitly imports the package `GUI`. `GUI:: Window` are therefore made visible to the contents of the package. The package `Server` does not import `GUI`. So, the contents of `Server` do not have permission to access any of the contents of `GUI`.
- The contents of `Client` have access to the exports of `Policies`, but they do not have access to the exports of `GUI`. To gain access, the `Client` would have to import GUI explicitly.

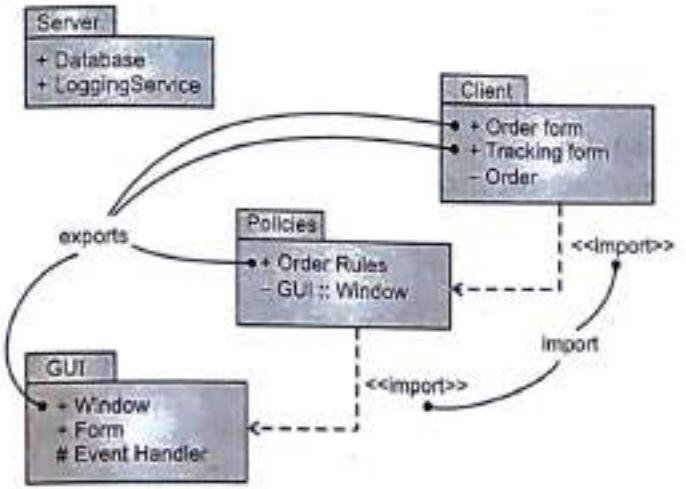


Fig. 5.55: Importing and Exporting in Package

5.11.5 Generalization

- Generalization among packages is similar to generalization among classes.
- Packages involved in generalization relationships follow the same principle of substitutability and do classes.
- Fig. 5.56 shows an example of Generalization.

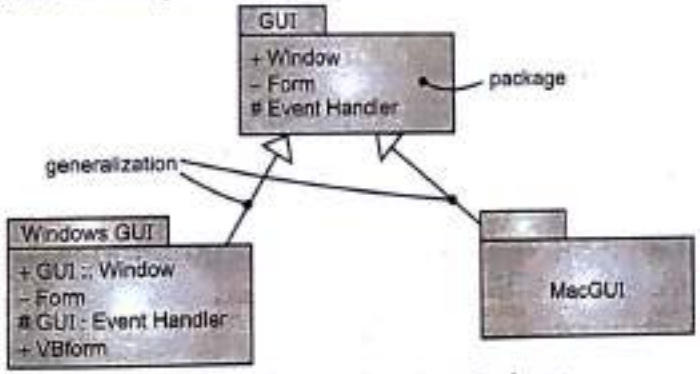


Fig. 5.56: Generalization among Packages

5.12 OBJECT DIAGRAM

- Object diagrams model the objects of things contained in class diagrams.
- Object diagrams are not only important for visualizing, specifying and documenting structural models, but also for constructing the static aspects of systems through Forward and Reverse Engineering.
- An object diagram in the Unified Modeling Language (UML) is a diagram that shows a complete or partial view of the structure of a modeled system at a specific time.
- In the Unified Modeling Language (UML), an object diagram focuses on some particular set of objects and attributes, and the links between these instances.
- Object diagram shows a snapshot of instances of things in class diagrams. Similar to class diagrams, object diagrams show the static design of a system but from the real or prototypical perspective.

Purpose of the Object Diagram: W-18, S-19

- Forward and reverse engineering.
- Object relationships of a system.
- Static view of an interaction.
- Understand object behaviour and their relationship from practical perspective.

5.12.1 Notations

- Basic Object Diagram Symbols and Notations of Object diagrams are given below:

Table 5.5: Notations (Symbols) used for Object Diagram

Name	Symbols	Description
1. **Class**	Class	A class describes a set of objects that share the same specifications of features, constraints, and semantics.
2. **Object names:**	Object name : Class *Named object* : Class *Unamed object* Object name : Class::Package *Named object with path name*	Each object is represented as a rectangle, which contains the name of the object and its class underlined and separated by a colon.

contd. ...

3. **Object attributes:**	Object name : Class Attribute type = 'Value' Attribute type = 'Value' Attribute type = 'Value' Attribute type = 'Value' *Object with attributes*	As with classes, we can list object attributes in a separate compartment. However, unlike classes, object attributes must have values assigned to them.
4. **Active object:**	Object name : Class *Active object*	Objects that control action flow are called active objects. Illustrate these objects with a thicker border.
5. **Multiplicity:**	Object name : Class *Multiple objects*	You can illustrate multiple objects as one symbol if the attributes of the individual objects are not important.
6. **Instance Specification:**	Instance	An instance specification is a model element that represents an instance in a modeled system. An instance specification specifies the existence of an entity in a modeled system and completely or partially describes the entity.
7. **Links:**	*Links*	Links are instances of associations. We can draw a link using the lines used in class diagrams. A link is a tuple with one value for each end of the association, where each value is an instance of the type of the end.
8. **Self-linked:**	Object name : Class	Objects that fulfill more than one role can be self-linked. For example, if Mark, an administrative assistant, also fulfilled the role of a marketing assistant, and the two positions are linked, Mark's instance of the two classes will be self-linked.

- Fig. 5.57 shows the basic concept of an Object diagram.

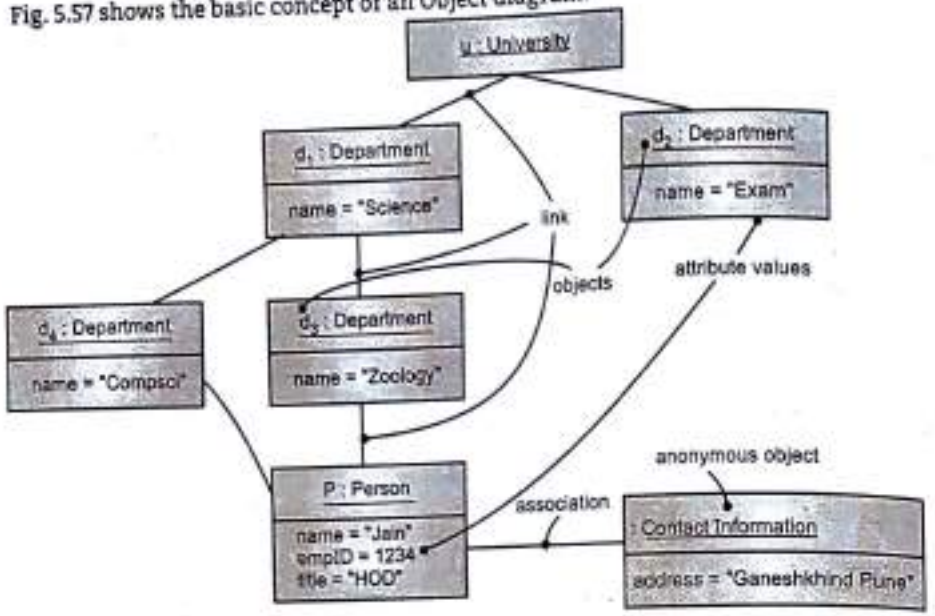


Fig. 5.57: Basic concept of an Object Diagram

5.12.2 Examples [S-18]

- Object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams.
- Object diagrams represent an instance of a class diagram. The basic concepts are similar for class diagrams and object diagrams.
- Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment.
- Object diagrams are used to render a set of objects and their relationships as an instance.

1. Object Diagram of an Order Management System:

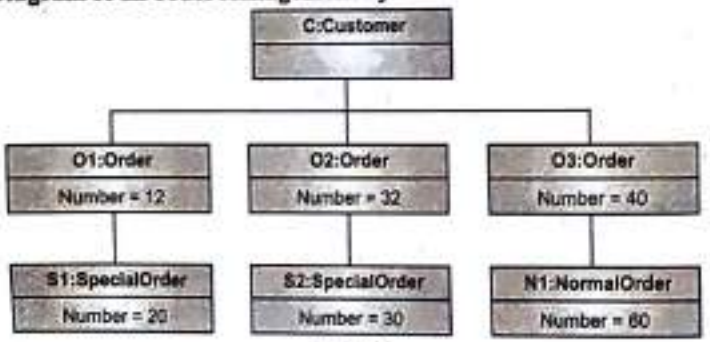


Fig. 5.58

2. An Object Diagram for Railway Reservation:

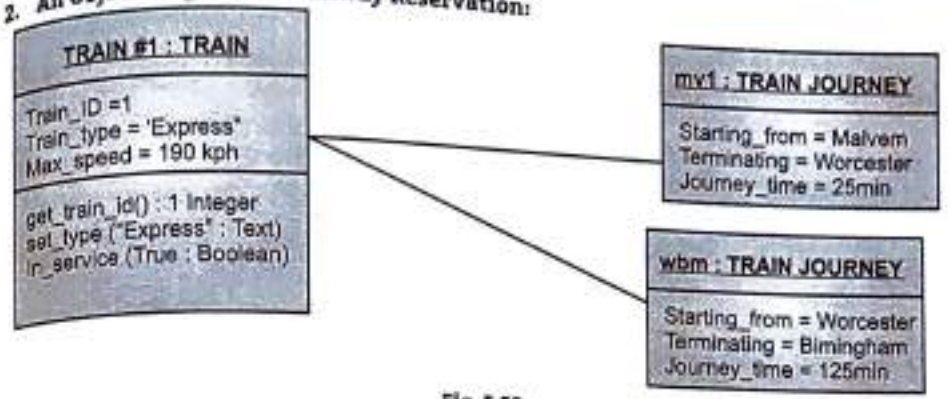


Fig. 5.59

3. Object Diagram for Hospital Management System:

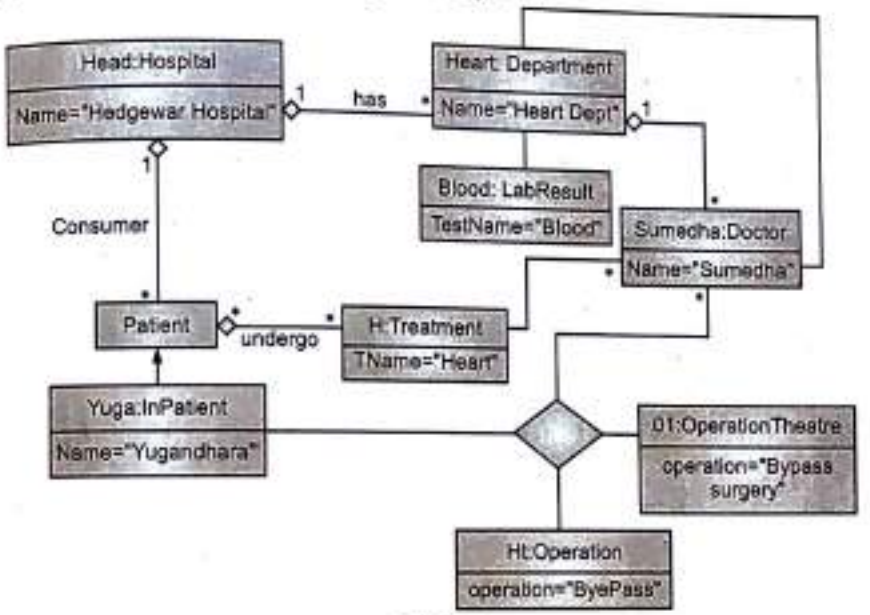


Fig. 5.60

Summary

- Structural modelling captures the static features of a system.
- Classes are the most important building blocks of any object-oriented system.
- A class acts as a logical entity that works as a template (blueprints) for creating similar kinds of objects.
- Instance of a class is called an object.

- An attribute is a named property of a class that describes a range of values that instances of the property may hold. A class may have any number of attributes.
- An operation is the implementation of a service that can be requested from any object of a class to affect behavior.
- In Object Oriented Modeling, there are three kinds of relationships, i.e. Dependency, Generalization and Association.
- Dependency in general means someone is dependent on another one.
- Dependency relationship is represented with a dashed directed line, directed towards the element which is dependent on.
- A generalization is a relationship between a general kind of thing (called the superclass or parent) and a more specific kind of thing (called the subclass or child).
- Generalization is a relationship between a superclass and a subclass which is a "is-a-kind of" relationship.
- An association is a structural relationship that specifies that objects of one thing are connected to objects of another.
- In many modeling situations, it is important for us to state how many objects may be connected across an instance of an association. This "how many" is called the multiplicity of an association's role.
- Aggregation is a special form of association.
- A whole/part type relationship will be represented with aggregation type of association relationship.
- Stereotype is an extension of basic building blocks which is similar one to existing one but more specific to our problem.
- A tagged value is a property of a stereotype, allowing us to create new information in an element bearing that stereotype.
- An interface is a collection of operations that are used to specify a service of a class or a component.
- Visibility indicates whether the attributes and operations of a classifier can be used by any other classifiers.
- There are three levels of visibility in UML used to specify the details of attributes and operations i.e., public, protected and private.
- Multiplicity is a specification of a range of allowable cardinalities an entity may assume.
- A package is a general purpose mechanism for organizing modeling elements into groups.
- An object diagram in the Unified Modeling Language (UML) is a diagram that shows a complete or partial view of the structure of a modeled system at a specific time.

Check Your Understanding

1. _____ is a blueprint for creating an object.
   (a) Class (b) Dependency
   (c) Generalization (d) Tagged values
2. Class is rendered as _____.
   (a) square (b) rectangle
   (c) circle (d) ellipse
3. CRC stands for _____.
   (a) Class Relationship Collaboration
   (b) Class Responsibility Component
   (c) Class Responsibility Collaboration
   (d) Collaboration Relation Component
4. Modeling simple collaboration are in _____ diagram.
   (a) class (b) object
   (c) use case (d) activity
5. A _____ is a connection among things.
   (a) class (b) relationship
   (c) stereotypes (d) tagged values
6. A dependency relationship is rendered as a _____.
   (a) ——— (b) --------➤
   (c) ———➤ (d) --------➤
7. A generalization relationship is rendered as a _____
   (a) ——— (b) --------➤
   (c) ———➤ (d) --------➤
8. _____ is rendered as a rectangle with a dog-eared corner.
   (a) Note (b) Stereotype
   (c) Tagged values (d) Constraints
9. Graphically, a stereotype is represented as a name enclosed by _____.
   (a) " " (b) [ ]
   (c) >> << (d) « »
10. Timing requirements are captured by using _____.
    (a) note (b) stereotype
    (c) tagged values (d) constraints

Answers

1. (a)	2. (b)	3. (c)	4. (a)	5. (b)	6. (b)	7. (c)	8. (a)	9. (d)	10. (d)

Practice Questions

Q.I Answer the following questions in short.

1. What is class? Explain with a labeled diagram.
2. Differentiate between link and association.
3. How is a note mechanism useful in UML? Explain with example.
4. Along with which type of relationship constraints mechanism can be denoted? Explain with an example.
5. Explain attributes with its syntax.
6. Consider a single object "Book" and draw object diagram with possible attributes.

Q.II Answer the following questions.

1. What is the relationship in UML? Explain dependency relationship.
2. Explain generalization type of relationship with example.
3. What is association relationship? Explain in detail the role, multiplicity and aggregation.
4. Explain what is extensibility mechanism? List out the terms and concepts under this.
5. What is the use of tagged values in UML diagram? Explain with an example.
6. What is the class diagram? Explain with an example.
7. Draw a class diagram for a departmental store system dealing in consumer items. Members enjoy facility and can purchase items as and when he or she needs them. Store has several counters and one can get almost all kinds of consumer items after visiting these counters.
8. Customer becomes member by paying initial membership amount and gets a credit card. Customers have to pay outstanding bills on quarterly basis. System sends reminders to the members. Defaulters not allowed purchasing items until the default amount is cleared. Consider all aspects of above mentioned problem and model diagram appropriately.
9. What is a classifier? List out different classifiers in UML with diagram.
10. Explain visibility modes along with a well labeled diagram.
11. Explain the concept of scope and specify its types with example.
12. Give an example to show abstract class, root, leaf and polymorphic elements.
13. Write syntax of operation and explain it with example.
14. Explain use of template classes along with example.
15. Explain dependency relationship along with stereotypes.
16. Explain possibilities of dependency relationship between different classifiers.
17. Explain generalization relationship along with stereotypes. Draw a diagram.
18. Explain association relationship with following:
    Navigation, Visibility, Qualification, Interface specifier, Composition, Association classes and Constraints.
19. What is realization relationship? Explain with an example.
20. What is package? Explain it with import and export stereotypes.
21. Explain generalization among packages.
22. What is an instance? Show different ways of instance representation in UML modeling.
23. Explain the use of object diagrams.

Q.III Define the terms:

1. Classifiers
2. Visibility
3. Abstract class
4. Root and leaf class
5. Polymorphic operation
6. Template classes
7. Interface
8. Package
9. Instance
10. Class
11. Attributes
12. Operations
13. Stereotype
14. Tagged value
15. Constraint
16. Note

Previous Exam Questions

Summer 2018

1. Define Generalization. [2 M]
   Ans. Refer to Section 5.3.2.
2. Consider a single object "customer" and draw object diagram with possible attributes. [2 M]
   Ans. Refer to Section 5.2.2.
3. What is dependency? [2 M]
   Ans. Refer to Section 5.3.1.
4. What is Association? Explain important terms in Association. [4 M]
   Ans. Refer to Section 5.3.3.
5. Draw class diagram for library management system. [4 M]
   Ans. Refer to Section 5.5.2.
6. What is package? Explain different kind of packages. [4 M]
   Ans. Refer to Section 5.11.
7. Explain dependency relationship along with different stereotypes. [4 M]
   Ans. Refer to Section 5.8.1.
8. Define Note: Refer to Section 5.4. [1 M]

Winter 2018

1. Define association [2 M]
   Ans. Refer to Section 5.3.3.
2. Define Tagged values. [2 M]
   Ans. Refer to Section 5.4.
3. What is Interface? [2 M]
   Ans. Refer to Section 5.9.
4. Write down the purpose of the object diagram. [2 M]
   Ans. Refer to Section 5.12.
5. Explain visibility modes along with well labelled diagrams. [4 M]
   Ans. Refer to Section 5.7.3.
6. Define Relationship. Explain different kinds of relationship. [4 M]
   Ans. Refer to Section 5.3.

7. Explain the concept of Aggregation with an example. [4 M]

Ans. Refer to Section 5.3.3.2.

8. Construct a design element for point of the sale terminal management system that can be used for buying and selling of goods in the retail shop. When the customer arrives at the post check point with the items to purchase, the cashier records each item price and add the item information to the running sales transaction. The description and price of the current items are displayed. On completion of the item entry the cashier informs the sales totals and tax to the customer. The customer chooses payment type (cash, cheque, credit/debit). After the payment is made the system generates a receipt and automatically updates the inventory, the cashier handovers the receipt to the customer.
Draw a class diagram. [4 M]

Ans. Refer to Section 5.5.2.

Summer 2019

1. What is Realization? [2 M]

Ans. Refer to Section 5.8.4.

2. Write down the purpose of the object diagram. [2 M]

Ans. Refer to Section 5.12.

3. Define class diagram. State the purpose of class diagram with example. [4 M]

Ans. Refer to Section 5.5

4. What is Package? Explain it with import and export stereotype. [4 M]

Ans. Refer to Section 5.11

5. What is classifier? List out different classifiers in UML with diagram. [4 M]

Ans. Refer to Section 5.7.1.

6. The case study titled Library Management System is Library management software for the purpose of monitoring and controlling the transactions in a library.
The case study on the library management system gives us the complete information about the library and the daily transactions done in a library. We need to maintain the record of new and retrieve the details of books available in the library which mainly focuses on basic operations in a library like adding new members, new books and up new information, searching books and members and facility to borrow and return books.
If features a familiar and well thought out, an attractive user interface, combined with strong searching insertion and reporting capabilities the report generation facility of library system helps to get a good idea of which are the books borrowed by the members, makes users possible to generate hard copy. [4 M]
Draw a Class diagram.

Ans. Refer to Section 5.5.2.

■■■

6... Basic Behavioural Modeling

Learning Objectives ...

- To understand concept of Behavioural Modeling.
- To understand the Use cases and Use case diagram.
- To know about Interaction diagram.
- To understand how to draw Sequence diagram and Activity diagram.

6.1 INTRODUCTION TO BASIC BEHAVIORAL MODELING

- We have learnt about the structural elements defined within the UML - things like classes, objects, interfaces, packages etc. and how structural views of the system could be constructed from assemblies of these elements. The other pillar of object-oriented modeling is the specification of dynamic behaviour.
- Behavior binds the structure of objects with their attributes and relationships so that the objects can meet their responsibilities.
- Object orientation focuses on how data and behavior relate under a single class. Emphasis on the object naturally obscured how objects are associated and how their behavior is defined in detail.
- Behavioral model describes the interaction in the system. In short, it represents the interaction among the structural diagrams.
- Dynamic or behavioral modeling emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects.
- Behavior modeling is the minimal form of logical abstraction that fits nicely into how the human mind thinks and allows for one to understand complex systems and essentially increasing one's own ability to solve difficult problems.
- Behavior diagrams show the dynamic behavior of the objects in a system, which can be described as a series of changes to the system over time.
- Behavioural model describes when the system is changing. Key feature is objects.
- All behavioural models describe the control structure of a system. This can be things like:
  1. Sequence of operations
  2. Object states
  3. Object interaction

6... Basic Behavioural Modeling

Learning Objectives ...

- To understand concept of Behavioural Modeling.
- To understand the Use cases and Use case diagram.
- To know about Interaction diagram.
- To understand how to draw Sequence diagram and Activity diagram.

6.1 INTRODUCTION TO BASIC BEHAVIORAL MODELING

- We have learnt about the structural elements defined within the UML - things like classes, objects, interfaces, packages etc. and how structural views of the system could be constructed from assemblies of these elements. The other pillar of object-oriented modeling is the specification of dynamic behaviour.
- Behavior binds the structure of objects with their attributes and relationships so that the objects can meet their responsibilities.
- Object orientation focuses on how data and behavior relate under a single class. Emphasis on the object naturally obscured how objects are associated and how their behavior is defined in detail.
- Behavioral model describes the interaction in the system. In short, it represents the interaction among the structural diagrams.
- Dynamic or behavioral modeling emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects.
- Behavior modeling is the minimal form of logical abstraction that fits nicely into how the human mind thinks and allows for one to understand complex systems and essentially increasing one's own ability to solve difficult problems.
- Behavior diagrams show the dynamic behavior of the objects in a system, which can be described as a series of changes to the system over time.
- Behavioural model describes when the system is changing. Key feature is objects.
- All behavioural models describe the control structure of a system. This can be things like:
  1. Sequence of operations
  2. Object states
  3. Object interaction

- We represent them with dynamic diagrams:
  - Sequence diagram.
  - Communication diagrams or Collaboration diagram.
  - State diagrams or State machine diagram or State chart.

6.1.1 Interactions

- When one object sends a message to another object and again another object tends to send the message to the next object this sequence of messages is nothing but a form of Interaction between objects. Objects can interact with each other by various ways like sending a message/ signal, invoking the operation or by passing some value.

6.2 USE CASES

- In this section, we discuss the basic concepts of use cases.

6.2.1 Overview of Use Cases

- A use case specifies the behavior of a system or a part of a system.
- Use case is a description of a set of sequences of actions including variants that a system performs to yield an observable result of value to an actor.
- Every interesting system interacts with human or automated actors that use that system for some purpose and those actors expect that system to behave in predictable ways.
- A use case represents a functional requirement of the system. A use case involves the interaction of actors and the system.
- An actor represents a coherent set of roles that users of use cases play when interacting with these use cases.
- An actor can be human or they can be automated systems.
- Use cases can be applied to the whole system. It can be applied to part of a system, including subsystems and even individual classes and interfaces.
- Graphically, a use case is rendered as an ellipse  as shown in Fig. 6.1.
- Fig. 6.1 shows use case Manage Account with Customer and Bank.

Fig. 6.1: Diagram for Manage Account with Customer and Bank

- Every use case must have a name that distinguishes it from other use cases. A name is a textual string. That name alone is known as a simple name; a qualified name is the use case name prefixed by the name of the package in which that use case lives.

- A use case diagram is typically drawn to show only its name, as in Fig. 6.2.

Fig. 6.2: Examples of use cases

6.2.2 Use Cases and Actors

- An actor is someone or something that must interact with the system under development.
- An actor represents a role that a human, a hardware device or even another system plays with a system.
- For example: In Fig. 6.3 LoanOfficer is an Actor for a bank and even a Customer is also Actor.

Fig. 6.3: Actor for a bank

- There are two types of Actors are as follows:
  1. **Primary Actor:** System delivers services after getting call from actor.
  2. **Secondary Actor:** System needs the help of actors for performing services called by the primary actor.
- Actors are used in models which are not actually part of the system but they live outside the system. We can define general kinds of actors (like Customers) and specialize them (such as Commercial Customers) using generalization relationships. Actors may be connected to use cases only by association.
- An association between an actor and a use case indicates that actor and the use case communicate with one another by sending and receiving messages as shown in Fig. 6.4.

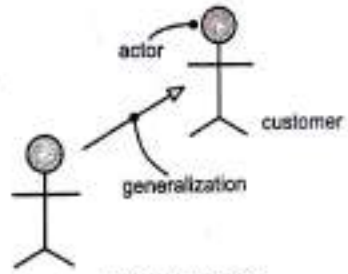


Fig. 6.4: Actors

6.2.3 Use Cases and Flow of Events

- A use case describes what a system, subsystem, class or interface does but it does not specify how it does it.
- Flow of events include how and when the use case starts and ends when the use case interacts with the actors and what objects are exchanged, and the basic flow and alternative flows of the behavior.
- The behavior of a use case can be specified by describing a flow of events in text.
- Flow of events is a step-by-step description of the interactions between the actor(s) and the system, and the functions that must be performed in the specified sequence to achieve a user goal.
- The most important part of a use case for generating test cases is the flow of events. The two main parts of the flow of events are:
  1. The basic flow of events should cover what "normally" happens when the use case is performed.
  2. The alternate flows of events cover behavior of an optional or exceptional character relative to normal behavior, and also variations of the normal behavior. You can think of the alternate flows of events as "detours" from the basic flow of events.
- Fig. 6.5 shows the basic flow of events and alternate flows of events for a use case.
- In Fig. 6.5, each of the different paths through a use case reflecting the basic and alternate flows are represented with the arrows.
- The basic flow, represented by the straight, back-line is the simplest path through the use case. Each alternate flow begins with the basic flow and then, dependent upon a specific condition, the alternate flow is executed.
- Alternate flows may rejoin the basic flow (Alternate Flows 1 and 3), may originate from another alternate flow (Alternate Flow 2), or may terminate the use case without rejoining a flow (Alternate Flows 2 and 4).

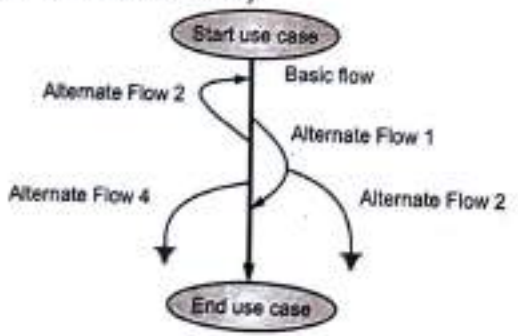


Fig. 6.5: Basic flow of events and alternate flows of events for a use case

6.2.4 Use Cases and Collaborations

- A use case is a description of sequences of actions that a system performs. A use case is used to structure the behavioural things in a model.
- A use case captures the intended behavior of the system i.e. class, subsystem or interface developing, without having to specify how that behavior is implemented.
- Use cases can be implemented by creating a society of classes and other elements that work together to implement the behavior of use cases.
- Collaborations are used to implement the behaviour of use cases with society of classes and other elements that work together. It includes static and dynamic structure.
- Realization of a use case can be specified by collaboration (See Fig. 6.6).

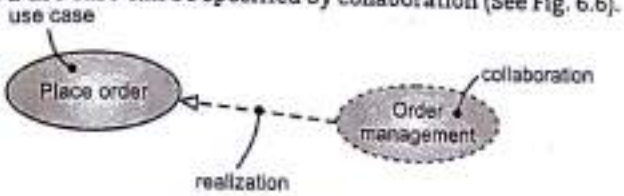


Fig. 6.6: Realization of a use case

6.2.5 Organising Use Cases [S-18]

- Use cases can be organized by grouping them in packages like classes.
- By using generalization include and extend relationship, we can organize use cases.
- It is a relationship between actors to support the reuse of common properties.
- A number of dependency types between use cases are:
  1. `<<extend>>` is used to include optional behavior from an extending use case in an extended use case.
  2. `<<include>>` is used to include common behavior from an included use case into a base use case in order to support re-use of common behavior.
- The generalization association is shown a solid line with an open arrowhead (or hollow triangle) as shown in Fig. 6.7.

1. **Include Relationship:**

- An include relationship between use cases means that the base use case explicitly incorporates the behavior of another use case at a location specified in the base.
- Include relationship is shown by include keyword having open and close triangles i.e. `<<include>>`.
- The relationship between a base use case and an included use case (See Fig. 6.7) is shown using a dashed line connecting the base use case to the included use case, with an open arrowhead pointing at the included use case. The line is labeled with the `<<include>>` stereotype.

2. Extend Relationship:

- An extend relationship between use cases means that the base use case implicitly incorporates the behavior of another use case at a location specified indirectly by the extending use case.
- Extend relationship is shown by extend keyword having open and close triangles i.e. `<<extend>>`.
- The relationship between a base use case and a use case that extends it (See Fig. 6.7) is shown using a dashed line connecting the extended use case to the base use case, with an open arrowhead pointing at the base use case. The line is labeled with the `<<include>>` stereotype.

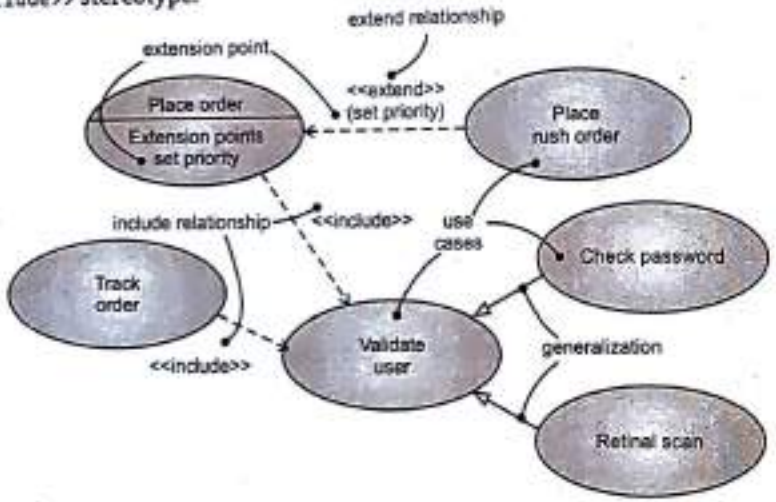


Fig. 6.7: The relationship between a base use case and a use case

6.2.6 Use Cases with Stereotypes

- Stereotypes are extensibility mechanisms in UML which allows designers to extend the vocabulary of UML in order to create new model elements.
- By applying appropriate stereotypes in your model, you can make the specification model comprehensible.
- Graphically, a stereotype is represented by a name enclosed by guillemets `<< >>`.
- As shown in the Fig. 6.8, the stereotype External User is applied to a model element (i.e. actor) called Customer.

Fig. 6.8: Stereotype External User

- Fig. 6.9 shows an example of stereotype.

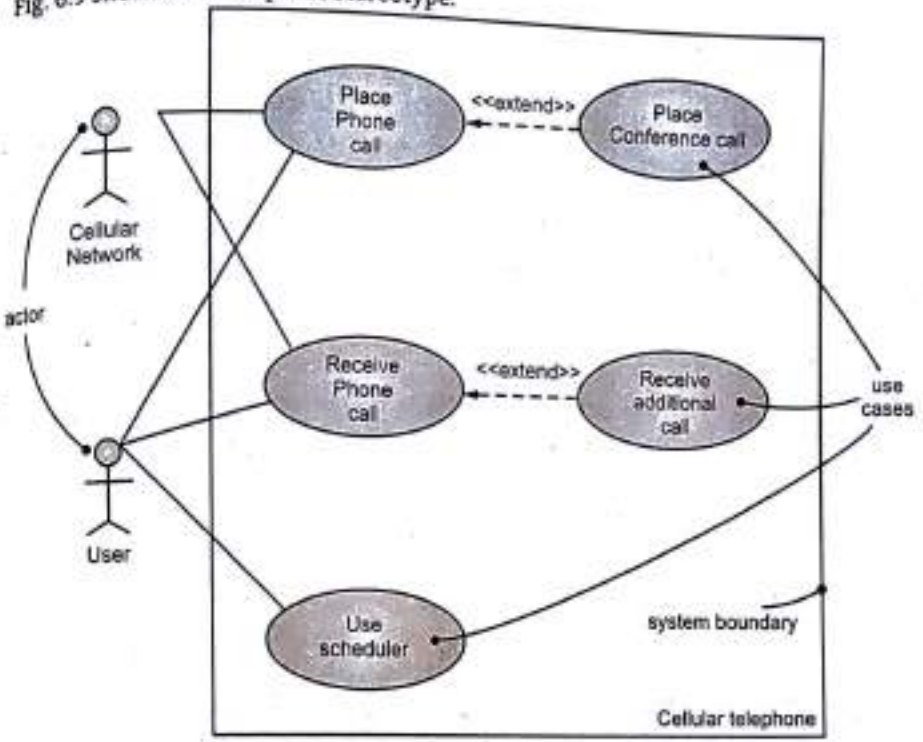


Fig. 6.9: Use case diagram

6.2.7 System Boundary

- System boundary shows how user interacts with the system. System is class in which use cases are executed.
- System boundary is a rectangle as shown in Fig. 6.10 and all use cases will be inserted inside this. Anchors will be outside this rectangle.

Fig. 6.10: System boundary

- **For example:** The actors appear outside the rectangle to indicate that they are not part of the system. Any external entity that interacts with the system, including another system, is an actor by definition. The name of the system or subsystem is usually shown inside the rectangle at the top, preceded by the stereotype `<<system>>` or `<<subsystem>>`.

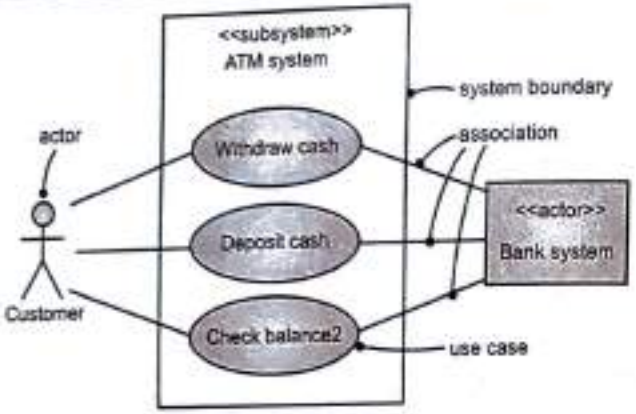


Fig. 6.11: Example of System Boundary

6.2.8 Communication Lines

- Use case diagrams have communication lines or link for communication between its various components.
- Use cases and actors have a certain relationship with one another. Relationships are modeled with lines.
- Linking actors and use cases in this way means that both communicate with each other.
- An actor is linked to use cases using simple association. This indicates an interaction with the system belonging to the use case, and in the context of that use case.
- Fig. 6.12 shows various communication lines in the use case diagram.
- Communication represents when one use case communicates information with another. Association shown as a solid line with no arrowhead. Association identifies an interaction between actors and use cases, (See Fig. 6.12).
- Generalizations are represented by a solid line with a solid arrowhead. Generalization defines a relationship between two actors or two use cases.
- Dependencies represented by dotted lines with arrowheads, (See Fig. 6.12). Dependency identifies a communication relationship between two use cases.

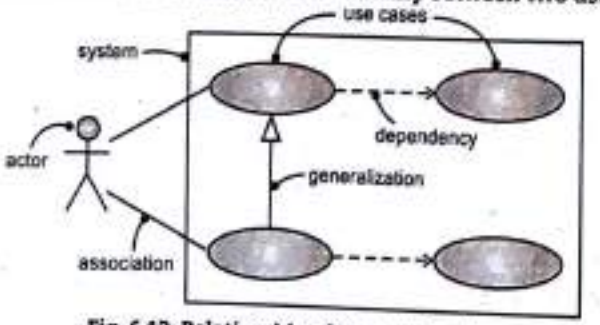


Fig. 6.12: Relationship of Communication

6.2.9 Relationships

- In use case diagrams, various types of relationships are used. In that relationship includes association, generalization, include and extend.
- UML has a simple way of indicating the relationships between actors and their use cases. You draw a line from each actor to each use case he or she (or it) participates in, (See Fig. 6.13).

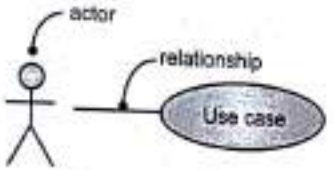


Fig. 6.13: Relationship

6.2.10 Multiplicity

- It can be useful to show the multiplicity of associations between actors and use cases. By this we mean how many instances of an actor interact with how many instances of the use case.
- By default we assume one instance of an actor interacts with one instance of a use case. In other cases we can label the multiplicity of one end of the association, either with a number to indicate how many instances are involved, or with a range separated by two periods (..). An asterisk (*) is used to indicate an arbitrary number.
- In the Fig. 6.14, you can clearly see that an instance of the Withdraw Cash use case may be initiated by one instance, and only one instance, of the Customer actor. Conversely, the Customer actor may initiate zero or one instance of the Withdraw Cash use case.

Fig. 6.14: Each end of an association may display multiplicity

6.3 USE CASE DIAGRAMS

- A use case diagram is a graphic representation of the interactions among the elements of a system.
- A use case diagram at its simplest is a representation of a user's interaction with the system and representing the specifications of a use case.
- A use case diagram can portray the different types of users of a system and the various ways that they interact with the system.
- A use case diagram is a diagram that shows a set of use cases and actors and their relationships.

- A use case diagram is just a special kind of diagram and shares the same common properties as do all other diagrams: a name and graphical contents that are a projection into a model.

Purpose of Use Case Diagrams:

1. Used to gather requirements of a system.
2. Used to get an outside view of a system.
3. Identify external and internal factors influencing the system.
4. Display the interacting among the requirements are actors.

- Use case diagrams commonly contain subject, use cases, actors, dependency, generalization and association relationships and so on.
- As Fig. 6.15 shows, use case diagram to model the behavior of the Airport Entry System.

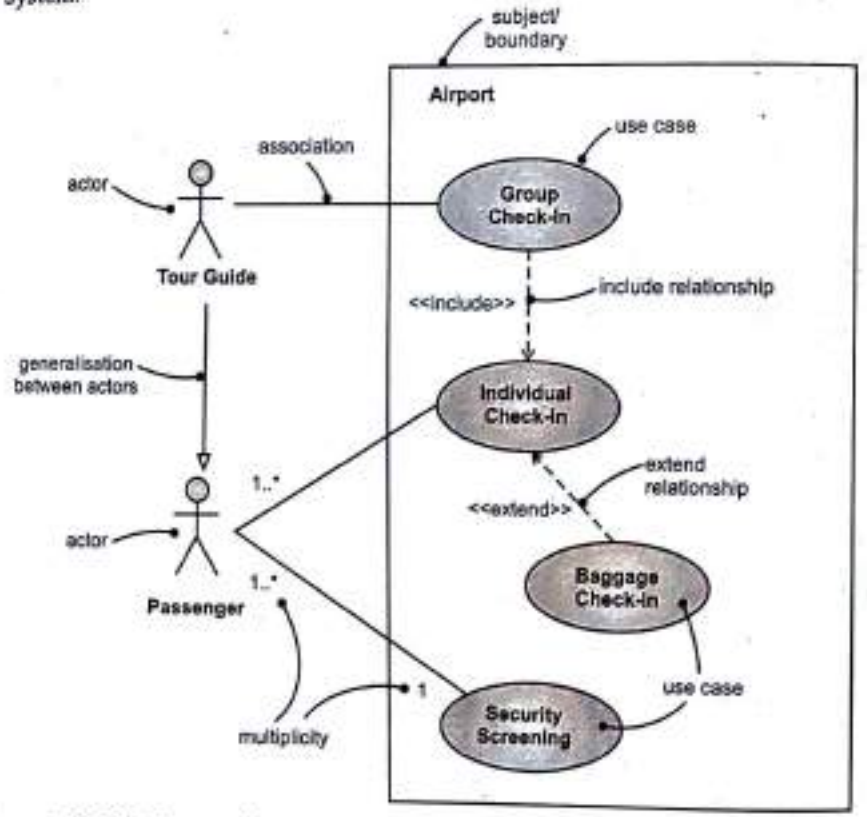
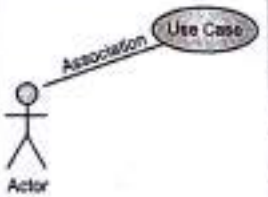
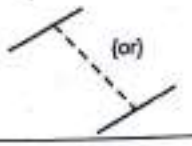


Fig. 6.15: Use case diagram to model the behavior of Airport Entry System

6.3.1 Notations

- Following table shows basic notations of use case diagrams.

Table 6.1: Basic notations of Use case Diagrams

Name	Symbols	Description
1. **Actor**	 Actor	An Actor models a type of role played by an entity that interacts with the subject, but which is external to the subject. Actors may represent roles played by human users, external hardware, or other subjects.
2. **Use Case**	 Use Case	A Use Case is the specification of a set of actions performed by a system, which yields an observable result that is a value for one or more actors or other stakeholders of the system.
6. **System/ Subject**	 System	If a subject or system boundary is displayed, the use case ellipse is visually located inside the system boundary rectangle.
4. **Association**	 Association Use Case Actor	An association specifies a semantic relationship that can occur between typed instances. It has at least two ends represented by properties, each of which is connected to the type of the end. More than one end of the association may have the same type.
5. **Collaboration**	 Collaboration	A collaboration specifies a view (or projection) of a set of cooperating classifiers. It describes the required links between instances that play the roles of the collaboration, as well as the features required of the classifiers that specify the participating instances.
6. **Constraint**	 (or)	Constraint is a condition or restriction expressed in natural language text or in a machine readable language for the purpose of declaring some of the semantics of an element.

contd. ...

7. Dependency		A dependency is a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation.
8. Extend		This relationship specifies that the behavior of a use case may be extended by the behavior of another (usually supplementary) use case. The extension takes place at one or more specific extension points defined in the extended use case.
9. Generalization		A generalization is a taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the more general classifier.
10. Include		It is a directed relationship between two use cases, implying that the behavior of the included use case is inserted into the behavior of the including use case.
11. Note		A Note (comment) gives the ability to attach various remarks to elements. A comment carries no semantic force, but may contain information that is useful to a modeler.
12. Realization		Realization is a specialized abstraction relationship between two sets of model elements, one representing a specification (the supplier) and the other represents an implementation of the latter (the client).

6.3.2 Examples

[S-18, W-18, S-19]

1. Use Case Diagram for ATM System:

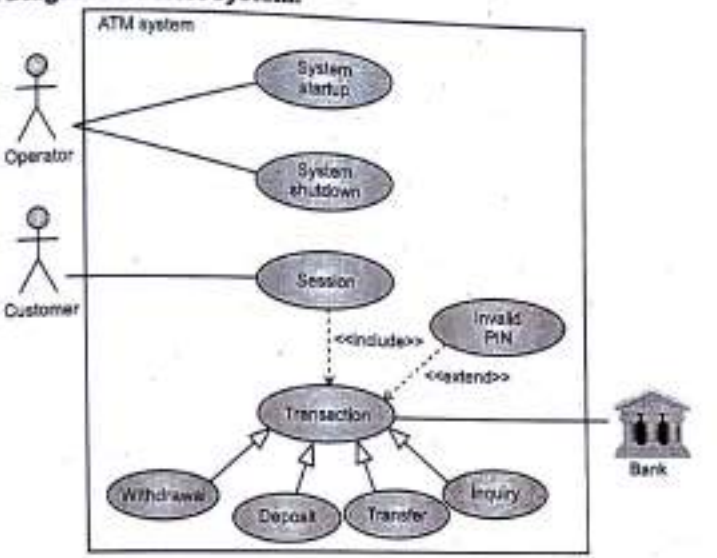


Fig. 6.16

2. Use Case Diagram for Online Shopping (Credit Cards Processing):

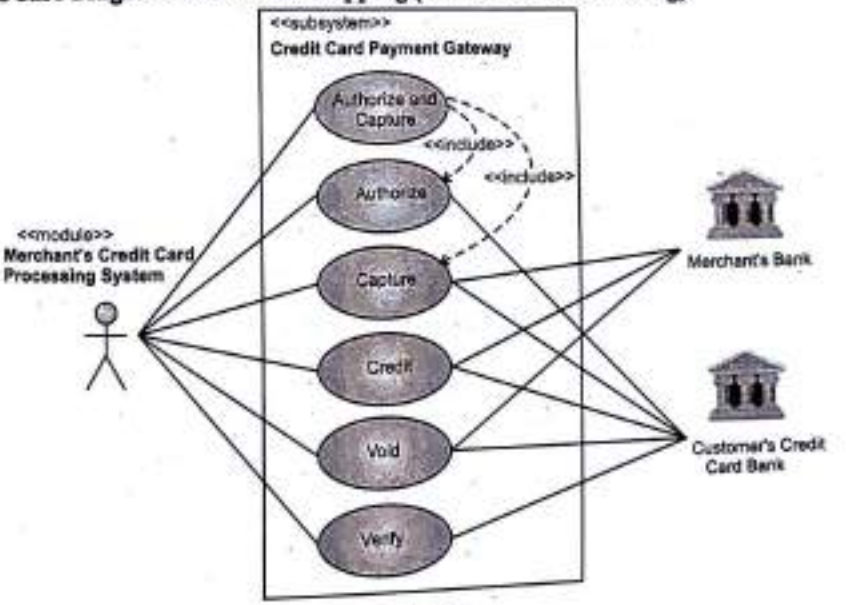


Fig. 6.17

3. Use Case Diagram for Library Management System:

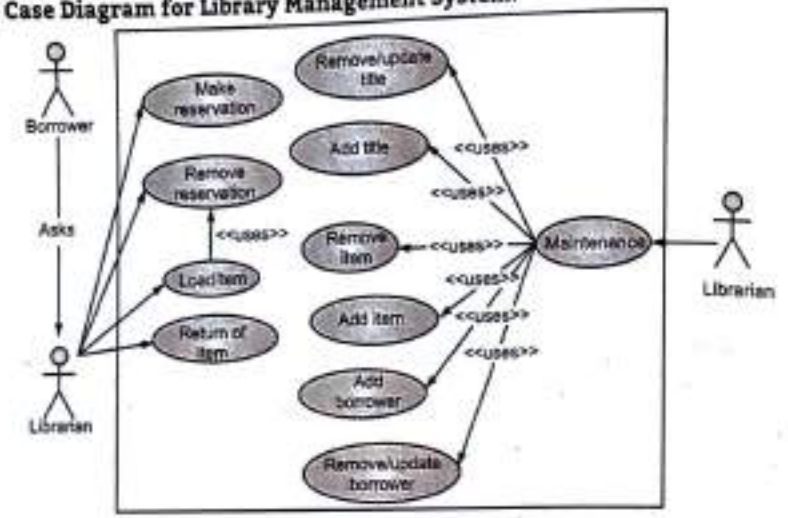


Fig. 6.18

4. Use Case Diagram for Railway Reservation System:

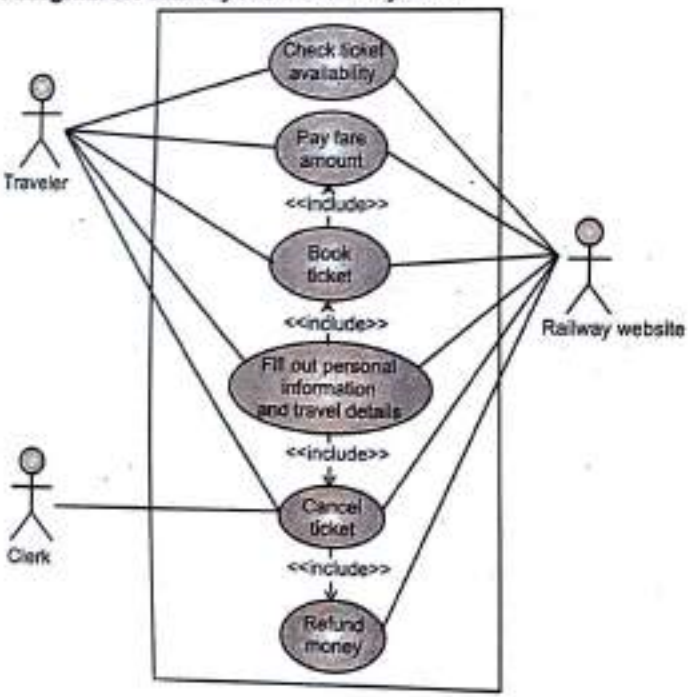


Fig. 6.19

5. Use Case diagram for Online Airline Reservation System:

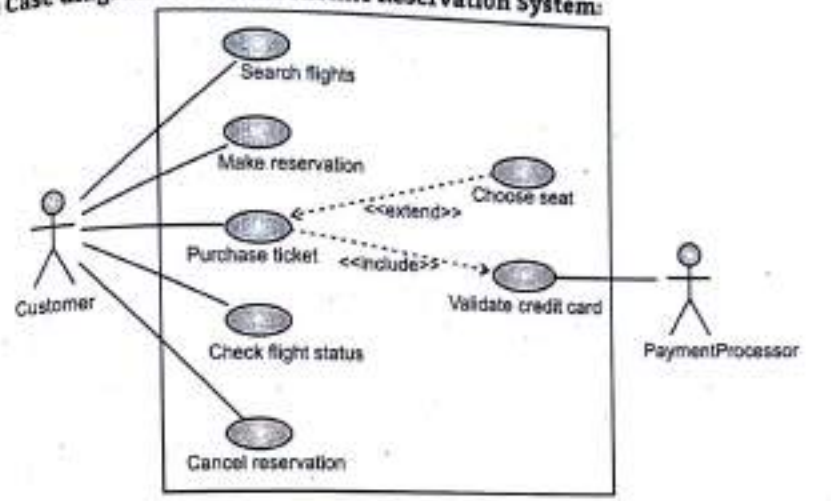


Fig. 6.20

6. Use Case Diagram for Hospital Management:

[S-19]

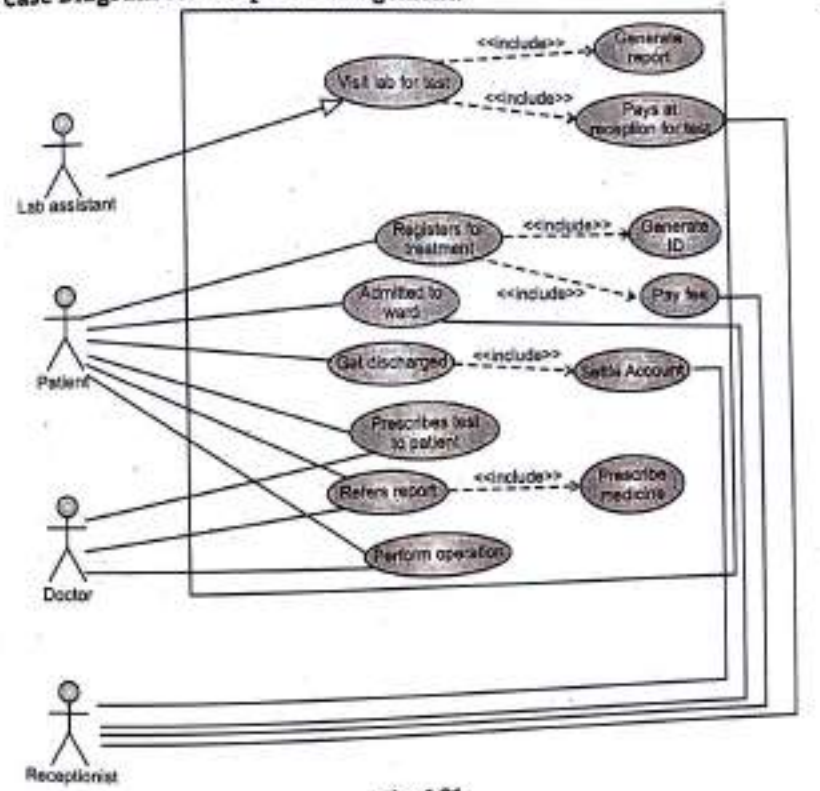


Fig. 6.21

6.4 INTERACTION DIAGRAMS [S-19]

- From the name Interaction, it is clear that the diagram is used to describe some type of interactions among the different elements in the model. So this interaction is a part of dynamic behaviour of the system.
- This interactive behaviour is represented in UML by two diagrams known as Sequence diagram and Collaboration diagram. The basic purposes of both the diagrams are similar.
- Sequence diagram emphasizes on time sequence of messages and collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.
- The purposes of interaction diagrams are to visualize the interactive behaviour of the system. Now visualizing interaction is a difficult task. So the solution is to use different types of models to capture the different aspects of the interaction.
- That is why sequence and collaboration diagrams are used to capture dynamic nature but from a different angle.

Purposes of Interaction Diagram:

1. To capture dynamic behaviour of a system.
2. To describe the message flow in the system.
3. To describe structural organization of the objects.
4. To describe interaction among objects.

6.4.1 Sequence Diagrams [S-18, S-19]

- A sequence diagram emphasizes the time ordering of messages.
- The sequence diagram will be formed by first placing the objects that participate in the interaction at the top of the diagram, along the x-axis. Place the object which initiates the interaction at the left and increasingly more subordinate objects to the right.
- A sequence diagram shows object interactions arranged in time sequence. It is a construct of a Message Sequence Chart.
- It represents the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.
- Sequence diagrams are typically associated with use case realizations in the logical view of the system under development.
- Sequence diagrams are sometimes called event diagrams or event scenarios.

6.4.1.1 Concept

- A sequence diagram is an interaction diagram. From the name it is clear that the diagram deals with some sequences, which are the sequence of messages flowing from one object to another.
- Interaction among the components of a system is very important from implementation and execution perspective.
- A sequence diagram is typically used to show object interactions in a single use case and it is easier to see the order in which activities occur.
- Major elements of the sequence diagram are shown in Fig. 6.22.

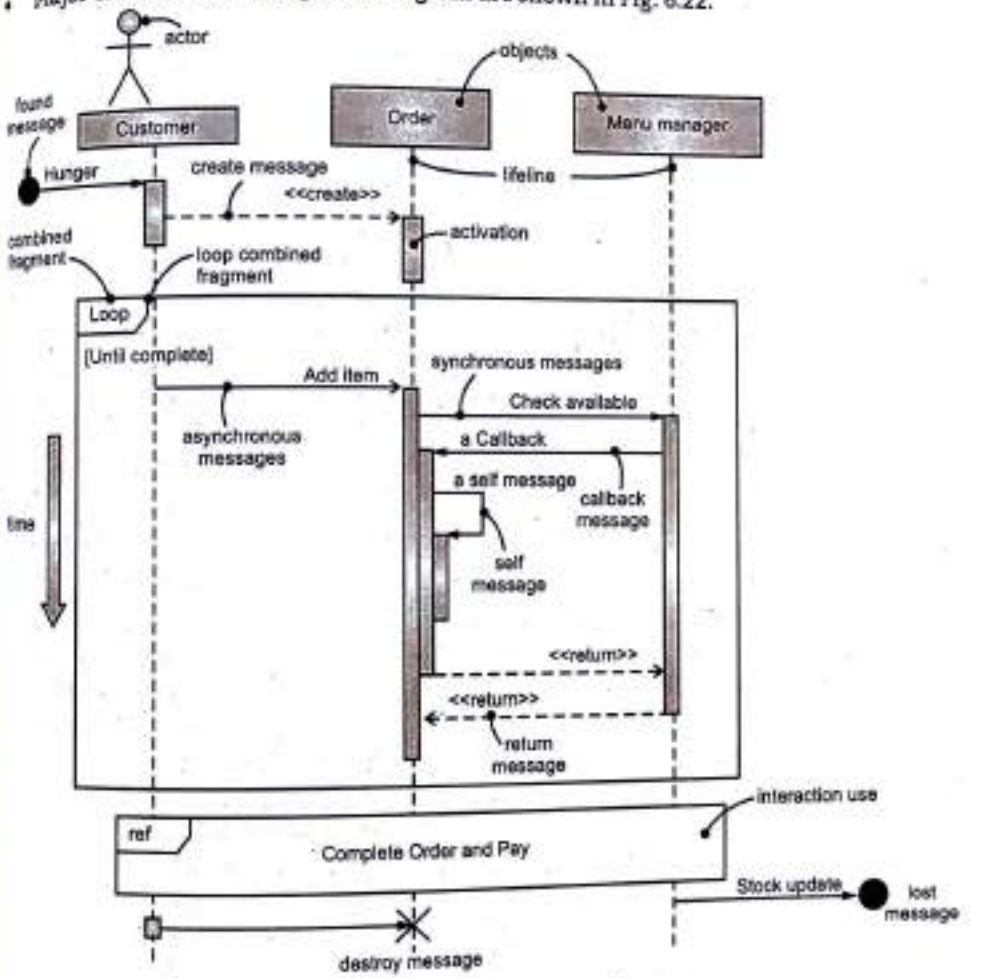
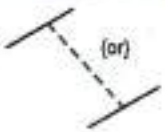


Fig. 6.22: Example of Sequence Diagram

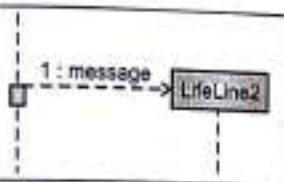
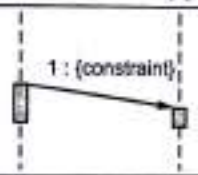
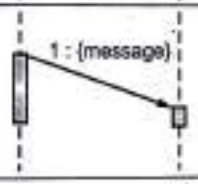
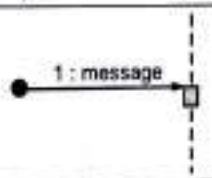
6.4.1.2 Notations

- Following table gives various notations used in sequence diagram.

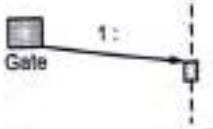
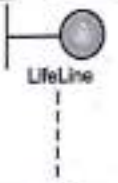
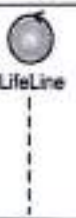
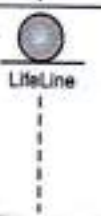
Table 6.2: Notations used in Sequence Diagram

Name	Symbols	Description
1. Actor		They model a type of role played by an entity that interacts with the subject, but which is external to the subject system.
2. Alternative Combined Fragment		A combined fragment defines an expression of interaction fragments. A combined fragment is defined by an interaction operator and corresponding interaction operands. Through the use of combined fragments the user will be able to describe a number of traces in a compact and concise manner. In short an alternative combined fragment represents a choice of behavior.
6. Call Message		A message defines a particular communication between lifelines of an interaction. Call message is a kind of message that represents an invocation of operation of the target lifeline.
4. Concurrent		A concurrent represents a session of concurrent method invocation along activation. It is placed on top of activation.
5. Constraint		A condition expressed in natural language text or in a machine readable language for the purpose of declaring some of the semantics of an element.

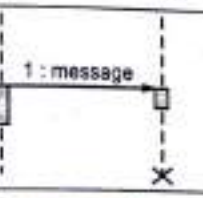
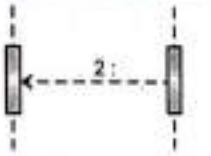
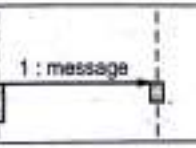
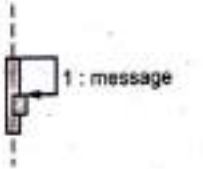
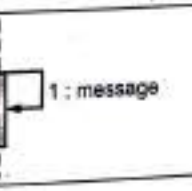
contd. ...

6. Continuation		A continuation is a syntactic way to define continuations of different branches of an alternative combined fragment. Continuation is intuitively similar to labels representing intermediate points in a flow of control.
7. Create Message		Create message is a kind of message that represents the instantiation of (target) lifeline.
8. Destroy Message		Destroy message is a kind of message that represents the request of destroying the life cycle of the target lifeline.
9. Duration Constraint		It defines a constraint that refers to a duration interval.
10. Duration Message		Duration message shows the distance between two time instants for a message invocation.
11. Found Message		A Found message is a message where the receiving event occurrence is known, but there is no (known) sending event occurrence.
12. Frame		A Frame represents an interaction, which is a unit of behavior that focuses on the observable exchange of information between connectable elements.

contd. ...

13. Gate		It is a connection point for relating a message outside an interaction fragment with a message inside the interaction fragment.
14. Interaction Use		It refers to an Interaction. The Interaction Use is shorthand for copying the contents of the referred interaction where the interaction use is.
15. LifeLine		A Lifeline represents an individual participant in the Interaction.
(i) LifeLine <<Boundary>>		It represents an individual participant in the Interaction.
(ii) LifeLine <<Control>>		It represents an individual participant in the Interaction.
(iii) LifeLine <<Entity>>		It represents an individual participant in the Interaction.
16. Loop Combined Fragment		A loop combined fragment represents a loop. The loop operand will be repeated a number of times.

contd. ...

17. Lost Message		It is a message where the sending event occurrence is known, but there is no receiving event occurrence. We interpret this to be because the message never reached its destination.
18. Message		A Message defines a particular communication between lifelines of an interaction.
19. Note		A Note (comment) gives the ability to attach various remarks to elements.
20. Return Message		Return message is a kind of message that represents the pass of information back to the caller of a corresponding former message.
21. Send Message		Send message is a kind of message that represents the start of execution.
22. Recursive Message		Recursive message is a kind of message that represents the invocation of a message of the same lifeline. Its target points to an activation on top of the activation where the message was invoked from.
23. Reentrant Message		A Reentrant message points to an activation on top of another activation.
24. Self Message		Self Message is a kind of message that represents the invocation of a message of the same lifeline.

6.4.1.3 Common terms/concepts used in Sequence diagram

- Let us see some important concepts in Sequence diagram:

1. Objects/Participants:

- Sequence diagram is an interaction diagram that shows the objects participating in a particular interaction and the messages they exchange/arrange in a time sequence.
- A sequence diagram is made up of a collection of participant's objects.
- Participants are the system parts that interact with each other during the sequence. In sequence diagram, the participant is an object or an entity.
- In sequence diagram, an object shown by a rectangle as shown in Fig. 6.23.

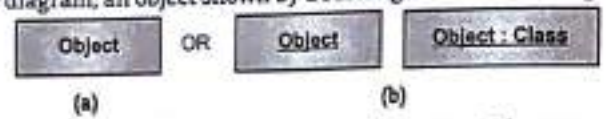


Fig. 6.23: Representation of Object in Sequence Diagram

- Fig. 6.24 shows an example of an object.

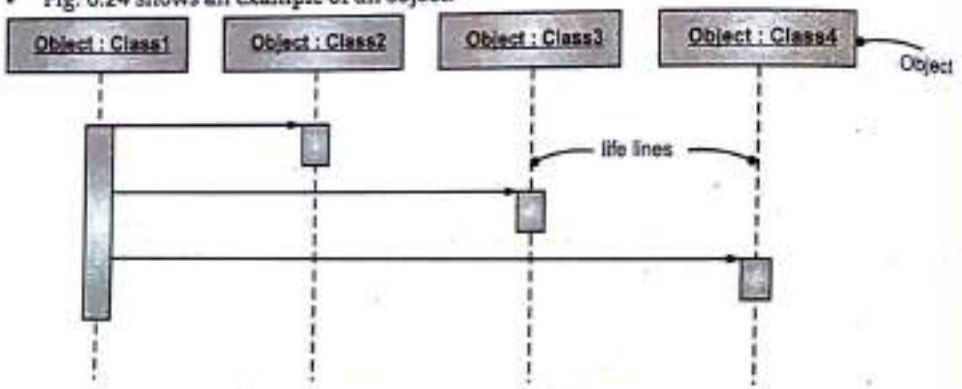


Fig. 6.24: Example of objects in Sequential diagram

Participant/Object Names:

- Participants on a sequence diagram can be named in number of different ways, picking elements from the following standard format:

```
name [selector]: class_name ref decomposition
```

- Fig. 6.25 shows participants/object name concept.

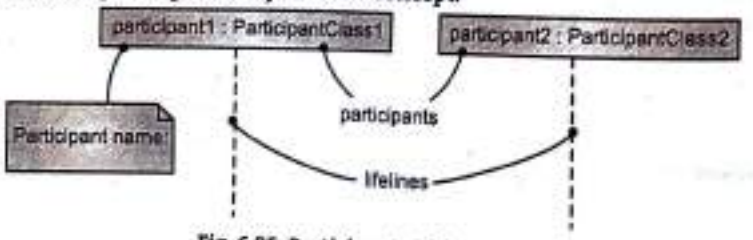


Fig. 6.25: Participants/object name

2. Activation Bars:

- When a message is passed to a participant, it invokes the receiving participant into doing something; at this point, the receiving participant is said to be active.
- To show that a participant is active, i.e. doing something you can use an activation bar as shown in Fig. 6.26.
- Activation bar represents the time an object needs to complete a task.

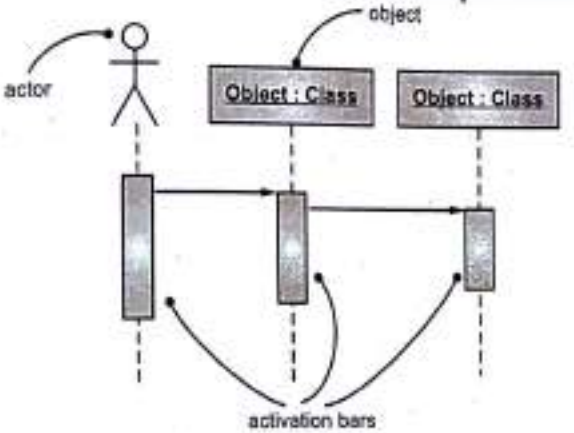


Fig. 6.26: Activation bars

3. Events:

- In a sequence diagram, the smallest part of an interaction is an event.
- An event is any point in an interaction where something occurs as shown on Fig. 6.27.

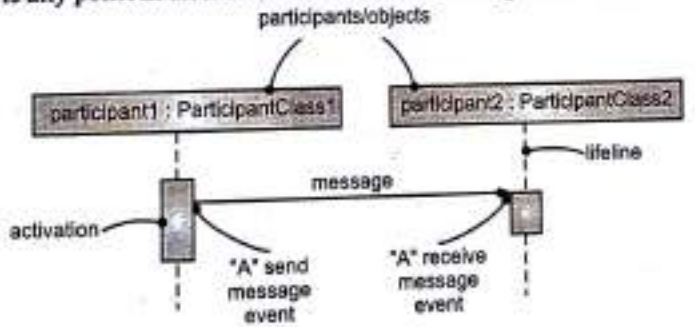


Fig. 6.27: Events

- Events are the building blocks for signals and messages. Signals and messages are really different names for the same concept, a signal is the terminology often used by system designers, while software designers often prefer messages.

4. Messages:

- An interaction in a sequence diagram occurs when one participant decides to send a message to another participant as shown in Fig. 6.28.

- Messages on a sequence diagram are specified using an arrow from the participant that wants to pass the message, the Message Caller, to the participant that is to receive the message, the Message Receiver.
- Messages can flow in whatever direction makes sense for the required interaction - from left to right, right to left.
- The message arrow comes with a description or signature. The format for a message signature is:

```
attribute * signal_or_message_name (arguments): return_type
```

- Messages are arrows that represent communication between objects as shown in Fig. 6.28.

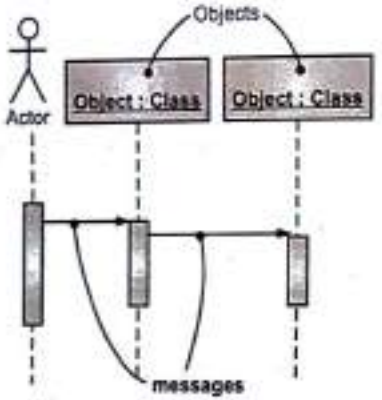


Fig. 6.28: Messages are arrows that represent communication between objects

Message Arrows:

- The type of arrowhead that is on a message is also important when understanding what type of message is being passed. For example, the Message Caller may want to wait for a message to return before carrying on with its work i.e., a synchronous message. Or it may wish to just send the message to the Message Receiver without waiting for any return as a form of "fire and forget" message i.e., an asynchronous message.
- Sequence diagrams need to show these different types of messages using various message arrows as shown in Fig. 6.29.

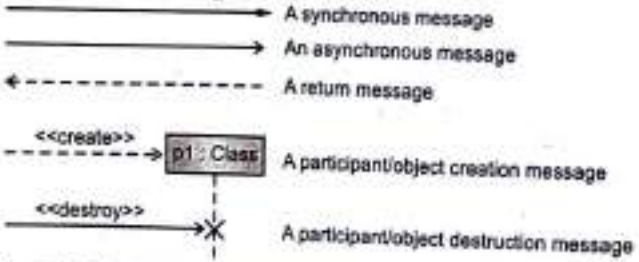


Fig. 6.29: Five types of message arrow for use on sequence diagram

5. **Lifelines:** S-18

- Lifelines are vertical dashed lines that indicate the object's presence over time.
- Lifelines are representative of roles/object instances that partake in the sequence being modeled.
- Fig. 6.30 shows an example of a lifeline.

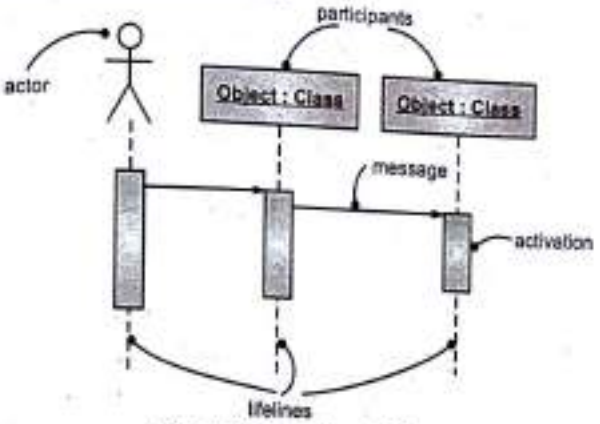


Fig. 6.30: Example of Lifeline

6.4.1.4 Examples

[S-18, W-18, S 19]

1. Sequence Diagram for Student Registration System:

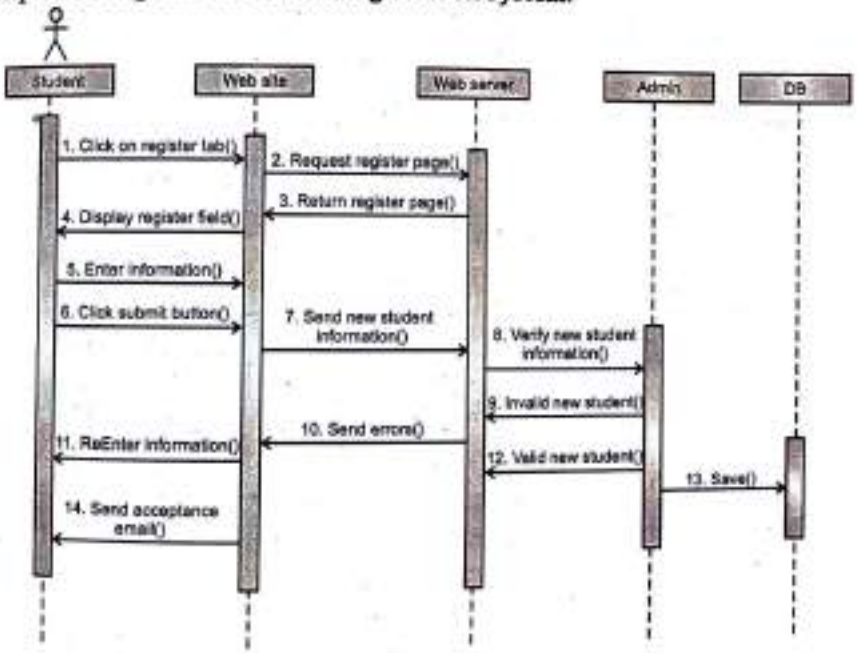


Fig. 6.31

2. Sequence Diagram for Library Management System:

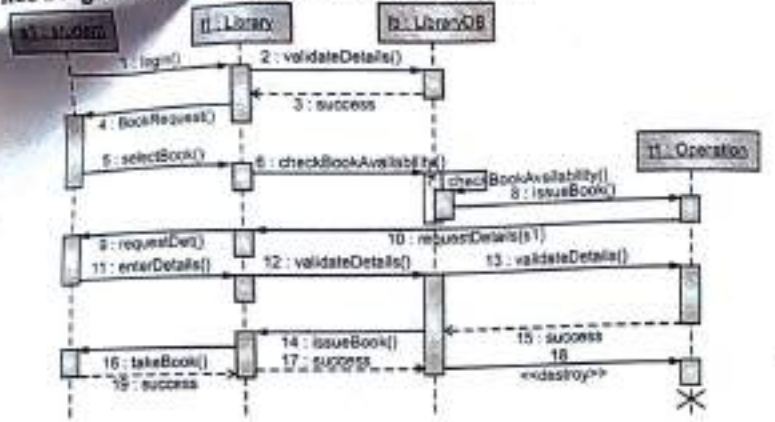


Fig. 6.32

3. Sequence Diagram for Hospital Management System:

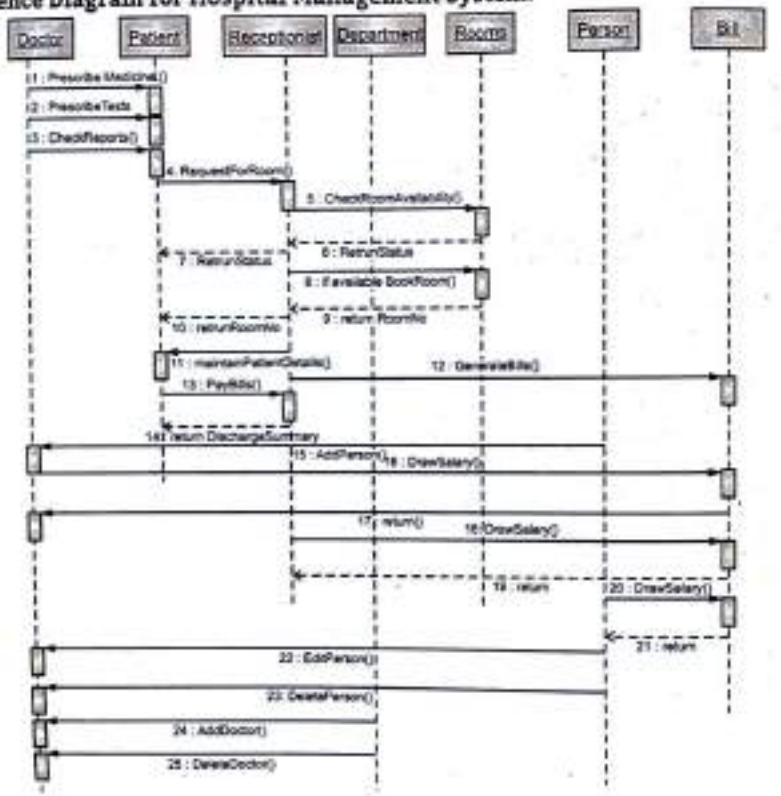


Fig. 6.33

4. Sequence Diagram for Railway Reservation System:

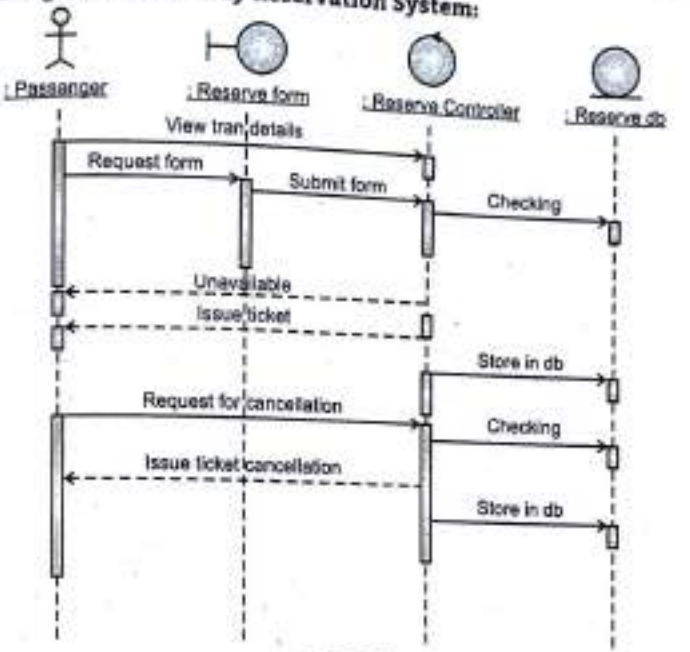


Fig. 6.34

5. Sequence Diagram for ATM Session:

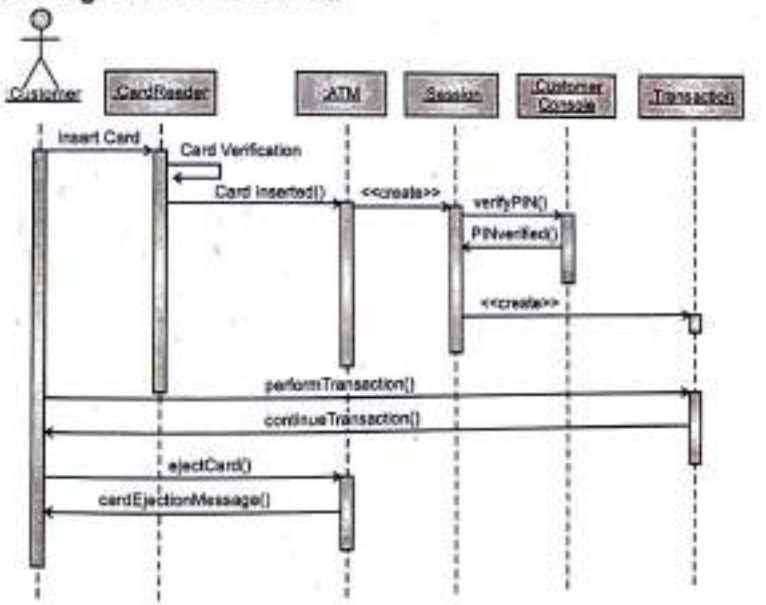


Fig. 6.35

6.5 ACTIVITY DIAGRAMS

- Activity diagrams are one of the five diagrams in the UML for modeling the dynamic aspects of systems.
- An activity diagram shows concurrency as well as branches of control.
- Activity diagrams may stand alone to visualize, specify, construct, and document the dynamics of a society of objects, or they may be used to model the flow of control of an operation.
- Activity diagrams consist of activities, states and transitions between activities and states.

6.5.1 Concept

- An activity diagram shows the flow from activity to activity. An activity is an ongoing non-atomic execution within a state machine.
- Activities ultimately result in some action, which is made up of executable atomic computations that result in a change in state of the system or the return of a value.
- Actions include calling another operation, sending a signal, creating or destroying an object, or some pure computation, such as evaluating an expression.
- Graphically, an activity diagram is a collection of vertices and arcs.

Purposes of Activity Diagrams:

1. Draw the activity flow of a system.
2. Describe the sequence from one activity to another.
3. Describe the parallel, branched and concurrent flow of the system.

- Activity diagrams commonly contain actions, activity nodes, flows, fork, join etc.
- We can model these dynamic aspects using activity diagrams, which focus first on the activities that take place among objects, as shown in Fig. 6.36.

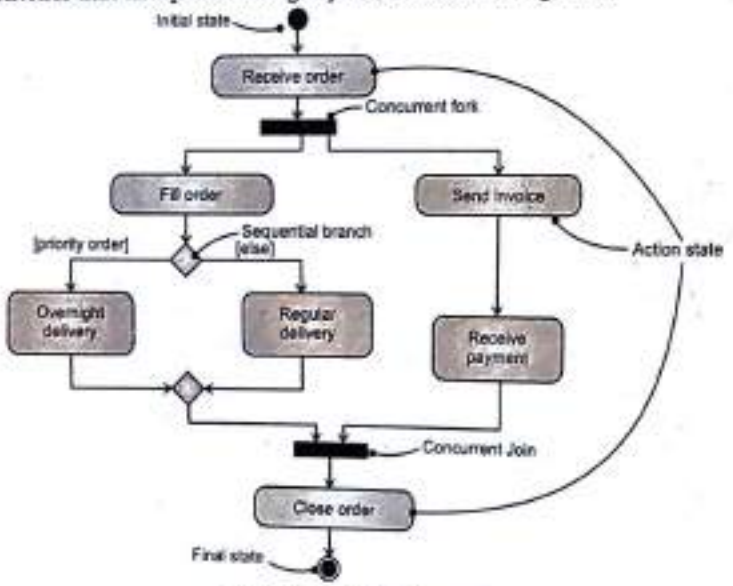
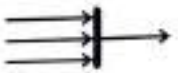


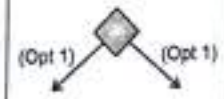
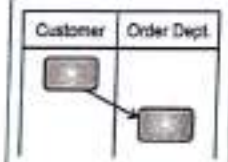
Fig. 6.36: Activity diagram

6.5.2 Notations

Table 6.3: Notations in Activity Diagram

Name	Symbol	Description
1. **Start/Initial Node**		It shows the starting point of the activity diagram. An initial or start node is depicted by a filled circle with black color.
2. **Final/Exit Node**		It shows the exit point of the activity diagram. An activity diagram can have zero or more activity final nodes. Final node is rendered as two concentric circles with filled inner circle.
3. **Action**		Actions are active steps in the completion of a process. Actions are denoted by rounded rectangles. Action is smallest unit of work which cannot be divided into further tasks.
4. **Activity**		Activity is parameterized behavior represented as coordinated flow of actions. An activity is the process being modeled, such as washing a car. An activity is a set of actions.
5. **Transition/ Edge/Path**		The flow of the activity is shown using arrowed lines called edges or paths. The arrowhead on an activity edge shows the direction of flow from one action to the next. A line going into a node is called an incoming edge, and a line exiting a node is called an outgoing edge.
6. **Fork Node**		It is used to show the parallel or concurrent actions. Steps that occur at the same time are said to occur concurrently or in parallel. Fork has a single incoming flow and multiple outgoing flows.
7. **Join Node**		The join means that all incoming actions must finish before the flow can proceed past the join. Join has multiple incoming flows and single outgoing flows.

contd. ...

8. Condition	[condition]	Condition text is placed next to a decision marker to let us know under what condition an activity flow should split off in that direction.
9. Decision/ Branch		A marker shaped like a diamond is the standard symbol for a decision. There are always at least two paths coming out of a decision and the condition text lets we know which options are mutually exclusive.
10. Note		A Note is used to render comments, constraints etc. of an UML element.
11. Swimlane		We use partitions to show which participant is responsible for which actions. Partitions divide the diagram into columns or rows (depending on the orientation of your activity diagram) and contain actions that are carried out by a responsible group. The columns or rows are sometimes referred to as swimlanes.
12. Flow Final Node		A Flow Final node terminates its own path, not the whole activity. The flow final node is represented as a circle with a cross inside.

6.5.3 Common terms used in Activity Diagrams

1. Action:

- Activity represents a behavior that is composed of individual elements that are actions.
- An action represents a single step within an activity.
- An action is an individual step within an activity, for example, a calculation step that is not deconstructed any further.

- Actions are denoted by round-cornered rectangles. Fig. 6.37 shows Process Order action.

Fig. 6.37: Process Order Action

- In the flow of control modeled by an activity diagram, things happen.
- We might evaluate some expression that sets the value of an attribute or that returns some value. Alternately, we might call an operation on an object, send a signal to an object, or even create or destroy an object. These executable, atomic computations are called actions.
- Fig. 6.38 shows inside the shape we may write an expression. Action states cannot be decomposed.
- Actions are atomic, i.e. events may occur, but the internal behavior of the action state is not visible. We cannot execute part of an action, either it executes completely or not at all.

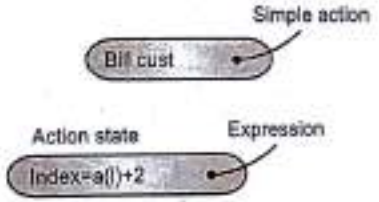


Fig. 6.38: Expression in Action

2. Activity Node:

- An activity is the specification of a parameterized sequence of behaviour.
- An activity node is an organizational unit within an activity.
- Activity nodes are nested groupings of actions or other nested activity nodes.
- Activity nodes have visible substructure.
- An activity diagram illustrates one individual activity.
- An activity is shown as a round-cornered rectangle enclosing all the actions, control flows and other elements that make up the activity.

Fig. 6.39: Notation of Activity

- We can think of an action state as a special case of an activity node. An action is an activity node that cannot be further decomposed. Similarly, we can think of an activity node as a composite, whose flow of control is made up of other activity nodes and actions.

- Fig. 6.40 shows, there is no notational distinction between action and activity states, except that an activity state may have additional parts, such as entry and exit actions.

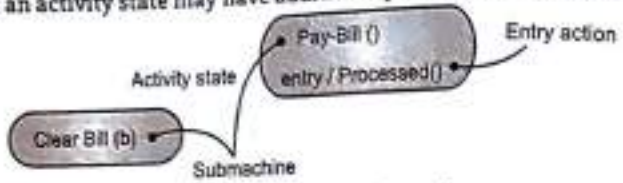


Fig. 6.40: Example of Activity nodes

3. Forking: S-18

- A fork represents the splitting of a single flow of control into two or more concurrent flows of control.
- A fork may have one incoming transition and two or more outgoing transitions, each of which represents an independent flow of control.
- Fig. 6.41 shows the fork, the activities associated with each of these paths continuing in parallel. Conceptually, the activities of each of these flows are truly parallel.
- We represent a branch using a diamond shape.

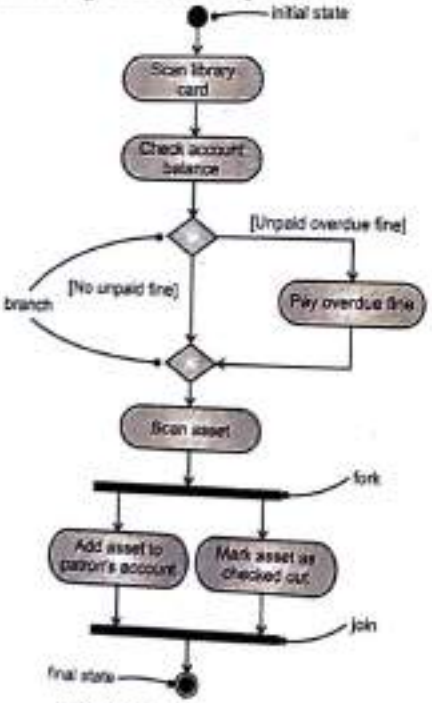


Fig. 6.41: Forking and Joining

4. Joining: S-18

- In Fig. 6.41, a join represents the synchronization of two or more concurrent flows of control.
- A join may have two or more incoming transitions and one outgoing transition. Above the join, the activities associated with each of these paths continue in parallel.
- At the join, the concurrent flows synchronize, meaning that each waits until all incoming flows have reached the join, at which point one flow of control continues below the join.
- Joins and forks should balance, meaning that the number of flows that leave a fork should match the number of flows that enter its corresponding join. Also, activities that are in parallel flows of control may communicate with one another by sending signals.

5. Swimlanes: S-19

- We will find it useful when we are modeling workflows of business processes, to partition the activity states on an activity diagram into groups, each group representing the business organization responsible for those activities.
- In the UML, each group is called a Swimlane because visually each group is divided from its neighbor by a vertical solid line, as shown in Fig. 6.42.

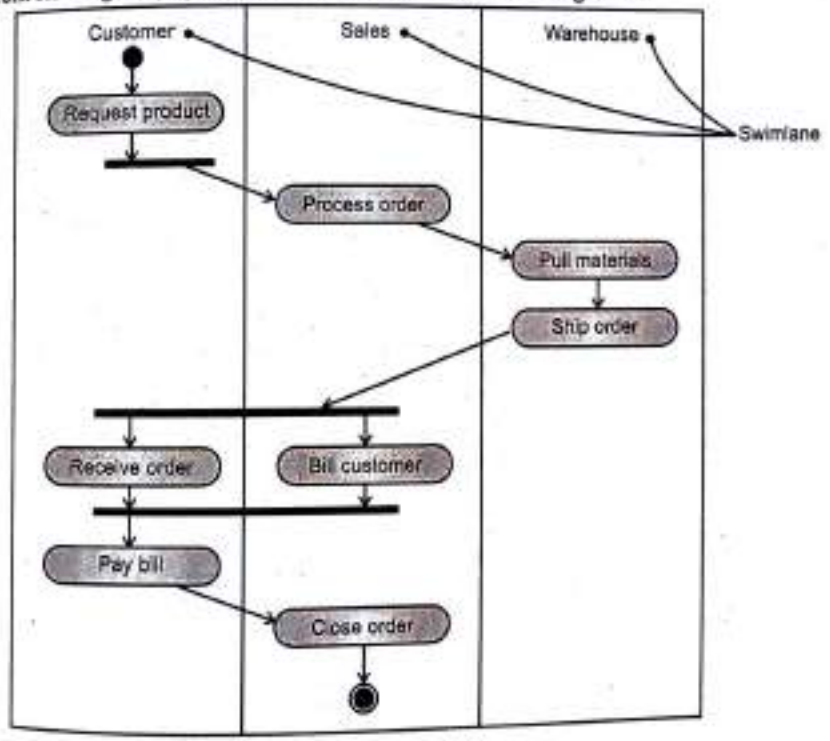


Fig. 6.42: Swimlanes

- A swimlane specifies a set of activities that share some organizational property.
- Each swimlane has a name unique within its diagram. It may represent some real-world entity, such as an organizational unit of a company.
- Each swimlane represents a high-level responsibility for part of the overall activity of an activity diagram, and each swimlane may eventually be implemented by one or more classes.
- In an activity diagram partitioned into swimlanes. Every activity belongs to exactly one swimlane but transitions may cross lanes.

6.5.4 Examples

[S-18, W-18, S-19]

1. Activity Diagram for ATM:

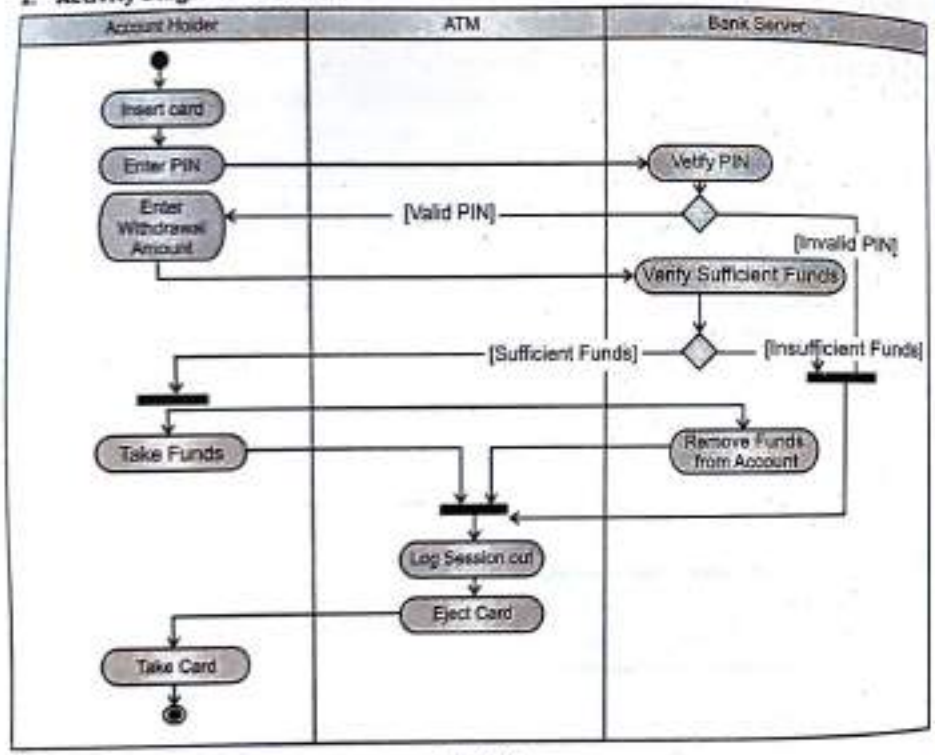


Fig. 6.43

1. Activity diagram for Hospital Management System:

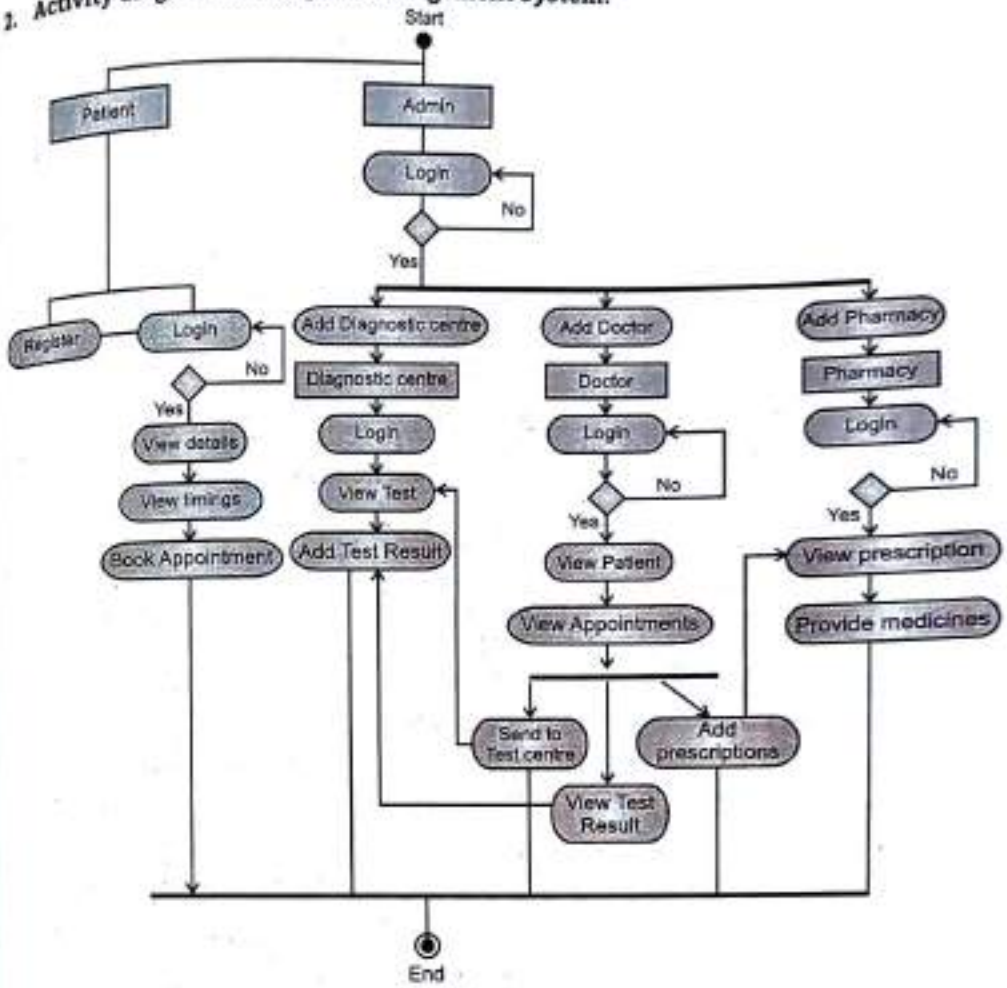


Fig. 6.44

3. Activity diagram for Library Management System:

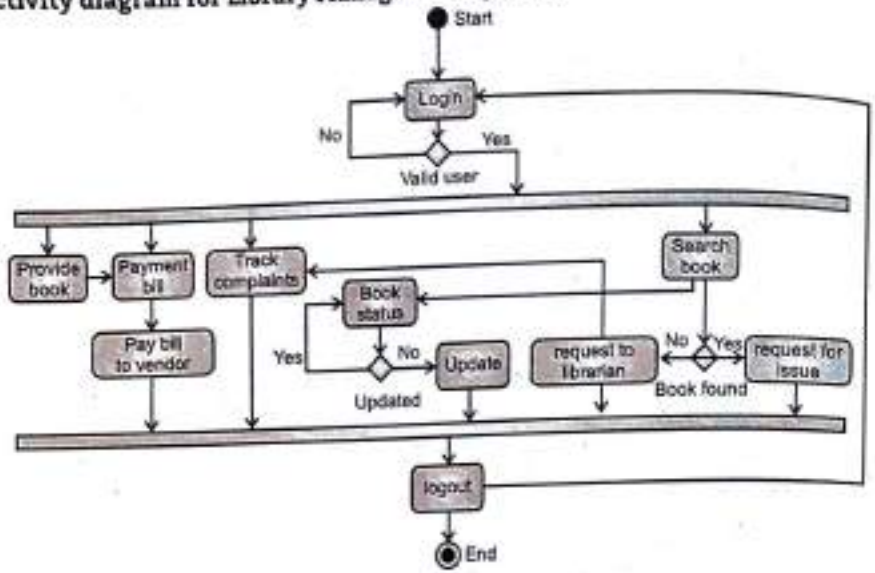


Fig. 6.45

4. Activity Diagram for Railway Reservation System

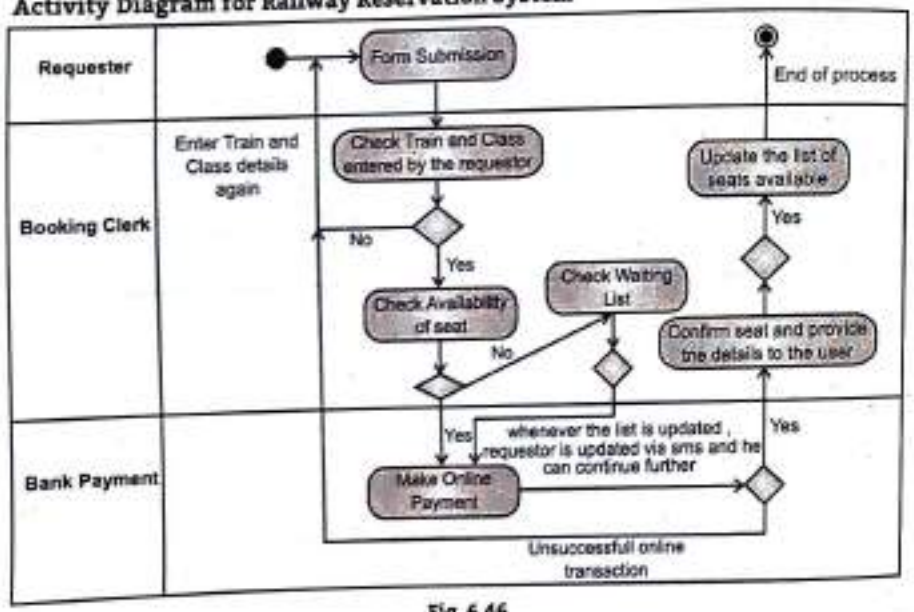


Fig. 6.46

5. Activity Diagram for College Management System:

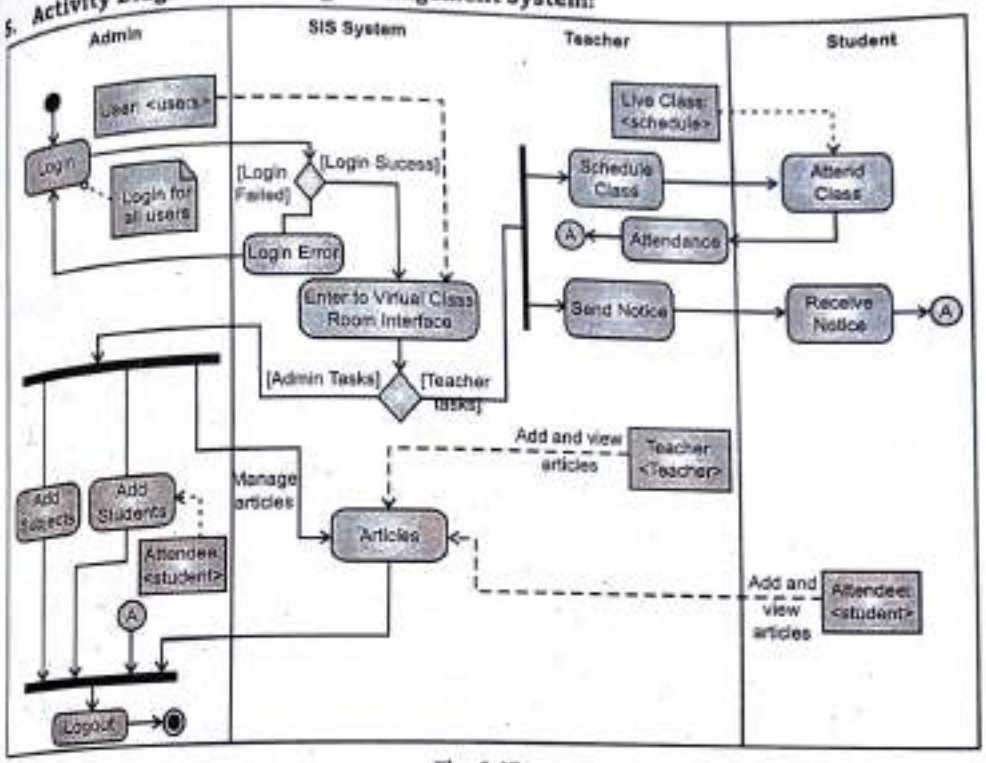


Fig. 6.47

6.6 STATE CHART DIAGRAMS

- Using an interaction, we can model the behavior of a group of objects that work together. Using a state machine, we can model the behavior of an individual object.
- A state machine is a behavior that specifies the sequences of states an object goes through during its lifetime in response to events together with its responses to those events.
- We use state machines to model the dynamic aspects of a system. We can specify the lifetime of the instances of a class, a use case, or an entire system. These instances may respond to such events as signals, operations, or the passing of time.
- When an event occurs, some effect will take place, depending on the current state of the object. An effect is the specification of a behavior execution within a state machine. Effects resolve in the execution of actions that change the state of an object or return values.

6.6.1 Concept

- State diagram itself clarifies the purpose of the diagram and other details. It describes different states of a component in a system. The states are specific to a component/object of a system.
- **Purpose of State Chart Diagram:**
  1. To model dynamic aspect of a system.
  2. To model life time of a reactive system.
  3. To describe different states of an object during its life time.
  4. Define a state machine to model states of an object.
- The UML provides a graphical representation of states, transitions, events, and effects or actions, for phone line as shown in Fig. 6.48.
- A state chart diagram shows a state machine, emphasizing the flow of control from state to state.
- A **state** of an object is a period of time during which it satisfies some condition, performs some activity, or waits for some event.
- A **transition** is a relationship between two states indicating that an object in the first state will perform certain actions and enter the second state when a specified event occurs and specified conditions are satisfied.
- An **event** is the specification of a significant occurrence that has a location in time and space. In the context of state machines, an event is an occurrence of a stimulus that can trigger a state transition.
- An activity is ongoing non-atomic execution within a state machine.
- An action is an executable atomic computation that results in a change in state of the model or the return of a value.
- Graphically, a state chart diagram is a collection of vertices and arcs.

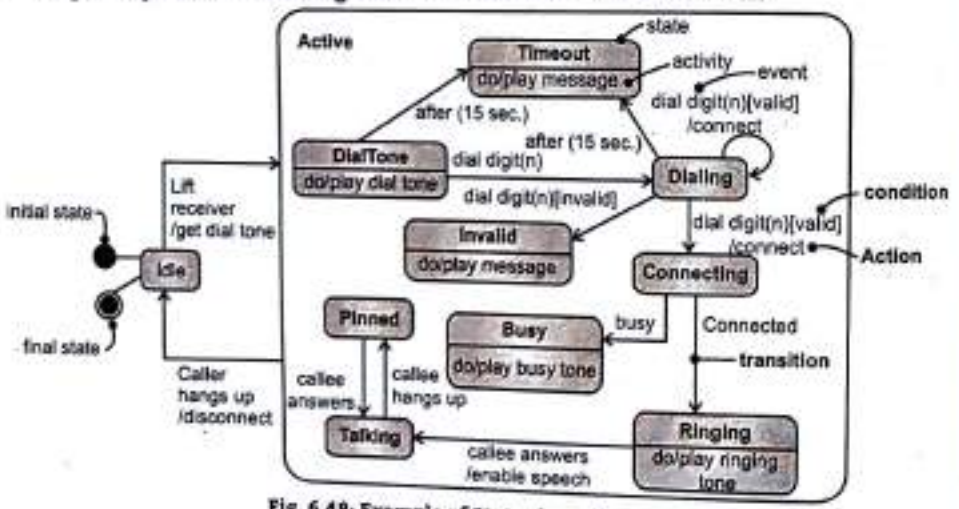


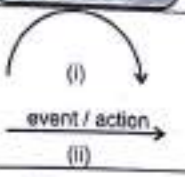
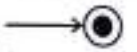
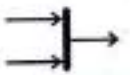
Fig. 6.48: Example of State chart diagram

6.6.2 Notations

S-18

- Following table gives various notation used in state diagram.

Table 6.4: Notation used in State Diagram

Name	Symbols	Description
1. State		States represent situations during the life of an object.
2. Transition		A solid arrow represents the path between different states of an object. Label the transition with the event that triggered it and the action that results from it.
3. Initial State		A filled circle followed by an arrow represents the object's initial state.
4. Final State		An arrow pointing to a filled circle nested inside another circle represents the object's final state.
5. Terminate		Entering a Terminate pseudo state implies that the execution of this state machine by means of its context object is terminated.
6. Constraint		A condition or restriction expressed in natural language text or in a machine readable language for the purpose of declaring some of the semantics of an element.
7. Junction		Junction vertices are semantic-free vertices that are used to chain together multiple transitions.
8. Join		Join vertices serve to merge several transitions emanating from source vertices in different orthogonal regions. The transitions entering a join vertex cannot have guards or triggers.
9. Fork		Fork vertices serve to split an incoming transition into two or more transitions terminating on orthogonal target vertices (i.e., vertices in different regions of a composite state). The segments outgoing from a fork vertex must not have guards or triggers.

6.6.3 Basic concepts of State Chart Diagram

1. States:

- A state is a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event.
- An object remains in a state for a finite amount of time. For example, a Heater in a home might be in any of four states: Idle (waiting for a command to start heating the house), Activating (its gas is on, but its waiting to come up to temperature), Active (its gas and blower are both on), and Shutting Down (its gas is off but its blower is on, flushing residual heat from the system).
- When an object's state machine is in a given state, the object is said to be in that state. For example, an instance of Heater might be Idle or perhaps ShutDown.
- A state has several parts:
  - (i) **Name:** A textual string that distinguishes the state from other states; a state may be anonymous, meaning that it has no name.
  - (ii) **Entry/exit actions:** Actions executed on entering and exiting the state, respectively.
  - (iii) **Internal transitions:** Transitions that are handled without causing a change in state.
  - (iv) **Substates:** The nested structure of a state, involving disjoint (sequentially active) or concurrent (concurrently active) substates.
  - (v) **Deferred events:** 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.

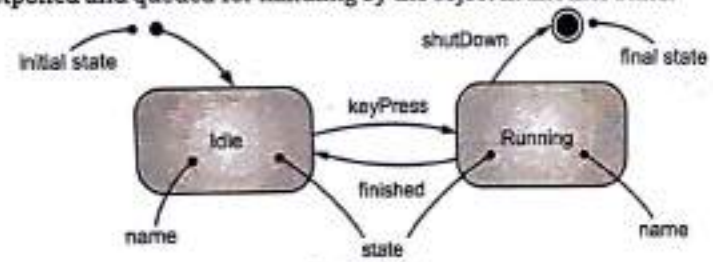


Fig. 6.49: States

2. Initial and Final States:

- In Fig. 6.50, there are two special states that may be defined for an object's state machine.
- First, there is the initial state, which indicates the default starting place for the state machine or substate.
- An initial state is represented as a filled black circle.
- Second, there is the final state, which indicates that the execution of the state machine or the enclosing state has been completed. A final state is represented as a filled black circle surrounded by an unfilled circle.

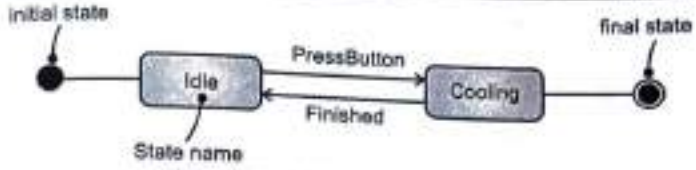


Fig. 6.50: Initial and Final States

3. Transitions:

- A transition is a relationship between two states indicating that an object in the first state will perform certain actions and enter the second state when a specified event occurs and specified conditions are satisfied.
- On such a change of state, the transition is said to be fire. Until the transition fires, the object is said to be in the source state; after it fires, it is said to be in the target state.
- For example, a Heater might transition from the Idle to the Activating state when an event such as too Cold (with the parameter desiredTemp) occurs.

Parts of Transitions:

- A transition has following five parts:
  - (i) **Source state:** The state affected by the transition; if an object is in the source state, an outgoing transition may fire when the object receives the trigger event of the transition and if the guard condition, if any, is satisfied.
  - (ii) **Event trigger:** The event whose reception by the object in the source state makes the transition eligible to fire, providing its guard condition is satisfied.
  - (iii) **Guard condition:** A Boolean expression that is evaluated when the transition is triggered by the reception of the event trigger. If the expression evaluates True, the transition is eligible to fire. If the expression evaluates False, the transition does not fire and if there is no other transition that could be triggered by that same event, the event is lost.
  - (iv) **Action:** An executable atomic computation that may directly act on the object that owns the state machine, and indirectly on other objects those are visible to the object.
  - (v) **Target state:** This state is active after the completion of the transition.
- A transition is rendered as a solid directed line from the source to the target state. A **self-transition** is a transition whose source and target states are the same.

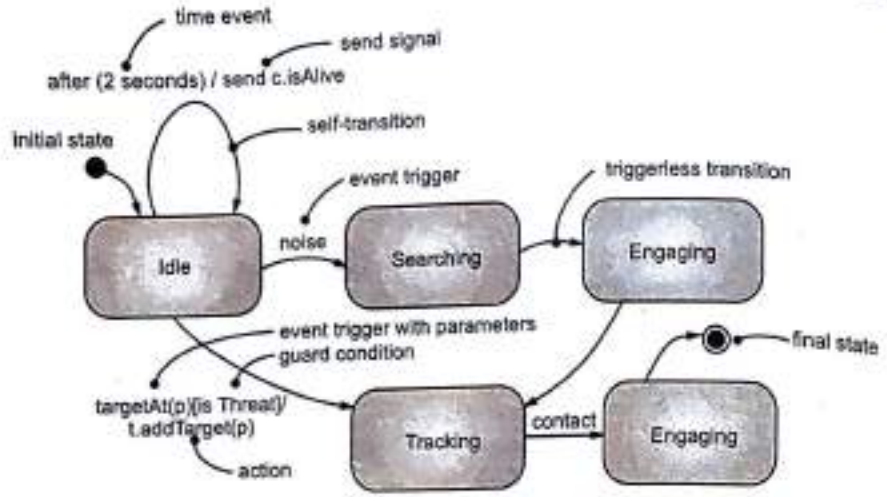


Fig. 6.51: Transitions

4. Substates:

- A substate is a state that is nested inside another one. For example, a Heater might be in the Heating state, but also while in the Heating state, there might be a nested state called Activating. In this case, it is proper to say that the object is both Heating and Activating.
- A state that has substates that is, nested states is called a **composite state**. A composite state may contain either concurrent (orthogonal) or sequential (disjoint) substates.
- In the UML, we render a composite state just as we do a simple state, but with an optional graphic compartment that shows a nested state machine. Substates may be nested to any level.

Example:

- Using sequential substates, there is a simpler way to model this problem, as Fig. 6.60 shows. Here the Active state has a substructure, containing the substates Validating, Selecting, Processing, and Printing.
- The state of the ATM changes from Idle to Active when the customer enters a credit card in the machine. On entering the Active state, the entry action readCard is performed.
- Starting with the initial state of the substructure, control passes to the Validating state, then to the Selecting state, and then to the Processing state.
- After Processing, control may return to selecting (if the customer has selected another transaction) or it may move on to printing.
- After Printing, there's a triggerless transition back to the Idle state. Notice that the Active state has an exit action, which ejects the customer's credit card.

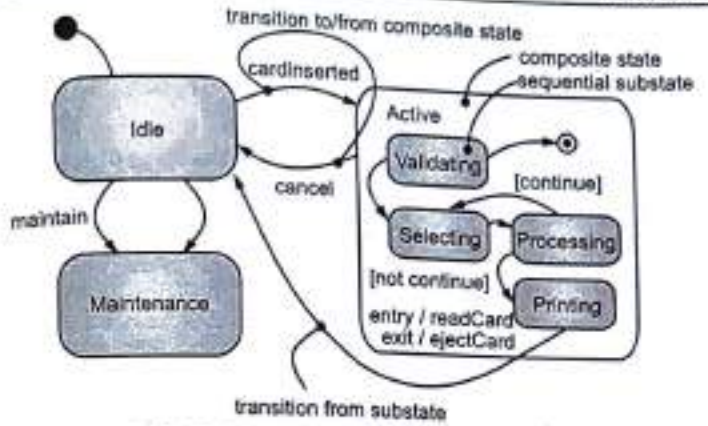


Fig. 6.52: Example of Sequential Substates

5. Activity:

- An activity is an operation that takes time to complete. An activity is associated with a state.
- Activities include continuous operations such as displaying a picture on a television screen as well as sequential operations that terminate by themselves after an interval of time such as closing valves.
- A state may control a continuous activity, such as ringing a telephone bell that persists until an event terminates it by causing a transition from state.
- The notation for activity is shown in Fig. 6.53.

Fig. 6.53: Notation of Activity

6. Event:

- An event is something that happens at a point in time such as user press a left button of mouse or a flight departing.
- An event is an occurrence, including the reception of a request.
- An event has no duration. One event may logically follow another or two events may be unrelated. An event is a one way transmission of information from one object to another.
- An object sending an event to another object may expect a reply, but the reply is a separate event under the control of the second object, which may or may not choose to send it.

Types of Events:

- There are a number of different types of events within the UML. Some of them are listed below:
  - **(i) CallEvent:** Associated with an operation of a class, this event is caused by a call to the given operation. The expected effect is that the steps of the operation will be executed.
  - **(ii) SignalEvent:** Associated with a signal, this event is caused by the signal being raised.
  - **(iii) TimeEvent:** An event caused by expiration of a timing deadline.
  - **(iv) ChangeEvent:** An event caused by a particular expression, (of attributes and associations) becoming true.

Syntax:

- An event is represented by its name. In the UML, an event is described using the following UML syntax:

```
event_name (parameter_list) [condition]
```

- Where,
  - `event_name`: Is the name of the event. An event usually has the same name as an operation of the element to which the state diagram pertains; therefore when the element receives the event, that operation is invoked.
  - `parameter_list`: Is optional, and is a comma-separated list that indicates the parameters passed to the event. Each parameter may be an explicit value or a variable. The parentheses are not used when the event does not require any parameters.
  - `condition`: Is optional, and indicates a condition that must be satisfied for the transition to fire or occur.

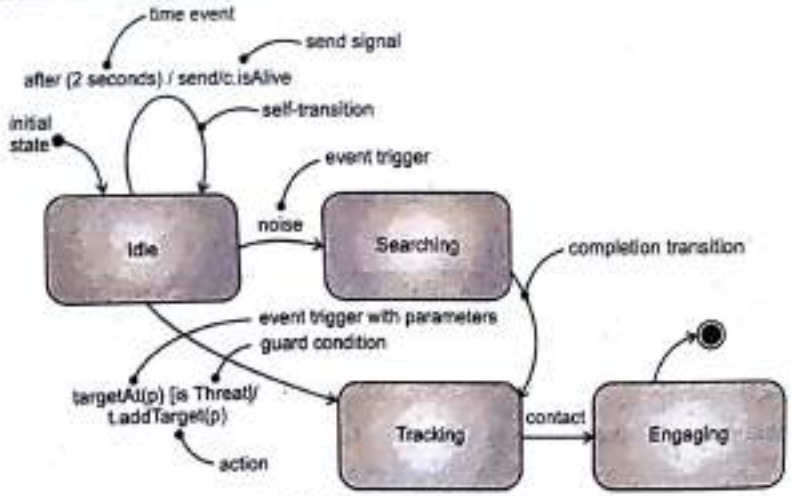


Fig. 6.54: Example of Events

6.6.4 Examples

1. State Chart Diagram for ATM:

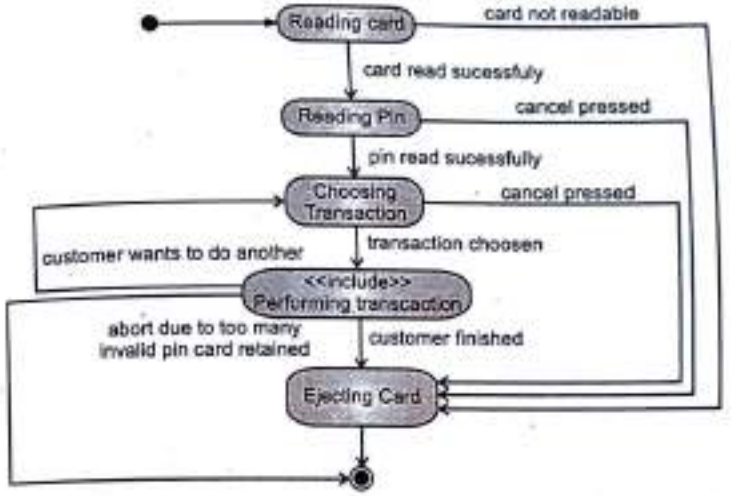


Fig. 6.55

2. State Chart Diagram of an Order Management System:

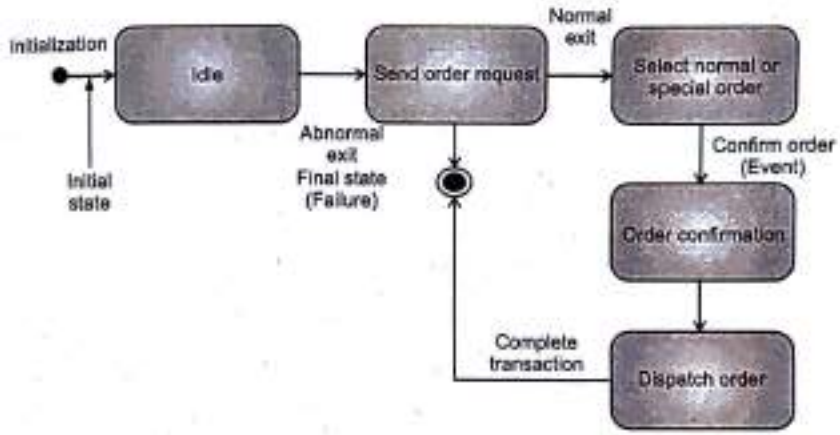


Fig. 6.56

3. State Chart Diagram Online Railway Reservation:

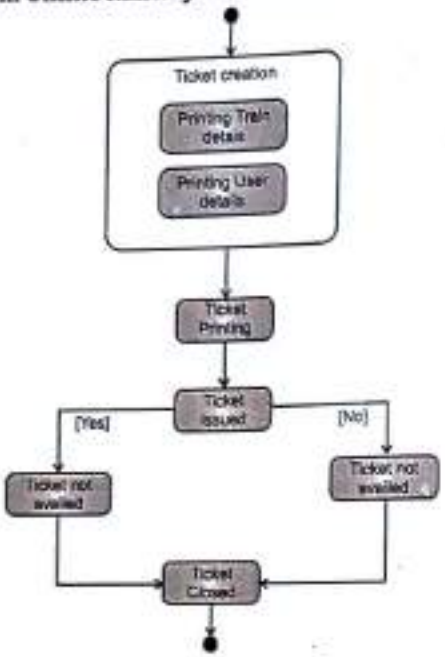


Fig. 6.57

4. State Chart Diagram for Passport Automation System:

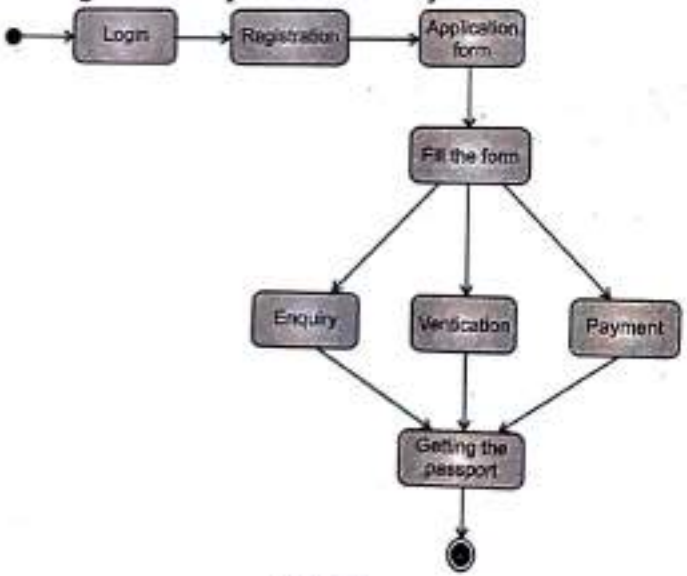


Fig. 6.58

5. State Chart Diagram for Library Management System:

(i) State diagram for Book:

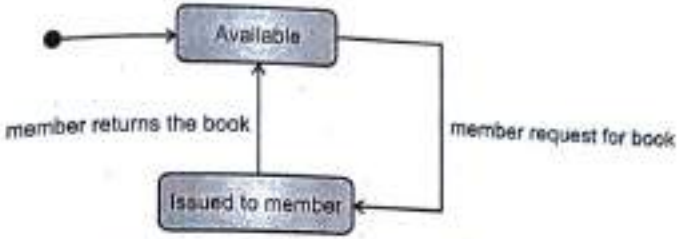


Fig. 6.59: State Diagram for Book Translation

(ii) State diagram for Librarian:

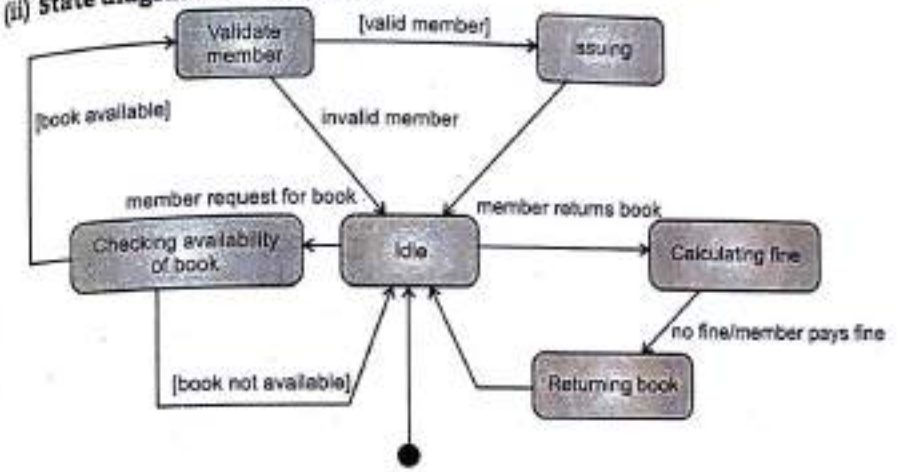


Fig. 6.60: State diagram for Librarian

Summary

- Behavior binds the structure of objects with their attributes and relationships so that the objects can meet their responsibilities.
- Dynamic or behavioral modeling emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects.
- Use case is a description of a set of sequences of actions including variants that a system performs to yield an observable result of value to an actor.
- An actor is someone or something that must interact with the system under development.
- Two types of dependency types between use cases are: Include and Extends.
- Sequence diagram emphasizes on time sequence of messages and collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

- Lifelines are vertical dashed lines that indicate the object's presence over time.
- An activity diagram is essentially a flowchart, showing flow of control from activity to activity. The activity can be described as an operation of the system.
- A state machine is a behavior that specifies the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events.
- A state chart diagram shows a state machine, emphasizing the flow of control from state to state.

Check Your Understanding

1. Use case descriptions consist of interaction _____.
   (a) Use case (b) Product
   (c) Actor (d) Product & Actor
2. Use case description consists of the following _____.
   (a) Actors (b) Number and Use case name
   (c) Need and stakeholder (d) All of the above
3. Select the true statement for the use case description format.
   (a) Underline text indicates to another use case
   (b) Extensions section utilize a complex numbering scheme
   (c) Indentation is used in a line to bring extensions easy to read
   (d) All of the above
4. Select the true in context to extensions.
   (a) The flow specifies the extensions
   (b) The alternatives are known as an extension because they extend the activity flow in various directions through the branch point
   (c) Both (a) and (b)
   (d) All of the above
5. What is the Interaction diagram?
   (a) Interaction diagrams are the UML notations for dynamic modeling of collaborations
   (b) Interaction diagrams are a central focus of engineering design
   (c) All of the mentioned
   (d) None of the mentioned
6. What is a sequence diagram?
   (a) A diagram that shows interacting individuals along the top of the diagram and messages passed among them arranged in temporal order down the page.
   (b) A diagram that shows messages superimposed on a diagram depicting collaborating individuals and the links among them.
   (c) A diagram that shows the change of an individual's state over time.
   (d) All of the mentioned

7. Interaction Diagram is a combined term for _____.
   (a) Sequence Diagram + Collaboration Diagram
   (b) Activity Diagram + State Chart Diagram
   (c) Deployment Diagram + Collaboration Diagram
   (d) None of the mentioned
8. Which of the following diagram is time-oriented?
   (a) Collaboration (b) Sequence
   (c) Activity (d) None of the mentioned
9. Who considers diagrams as a type of Class diagram, Component diagram, Object diagram, and Deployment diagram?
   (a) Structural (b) Behavioral
   (c) Non-behavioral (d) Non-structural
10. Which type they considered Activity diagram, Use case diagram, Collaboration diagram, and Sequence diagram?
    (a) Non-behavioral (b) Non-structural
    (c) Structural (d) Behavioral

Answers

1. (d)	2. (d)	3. (d)	4. (d)	5. (c)	6. (a)	7. (a)	8. (b)	9. (a)	10. (d)

Practice Questions

Q.I Answer the following questions in short.

1. What is the use of state chart diagram?
2. What is meant by Architectural Modeling?
3. What is meant by component diagrams?
4. Enlist various types of components.
5. Explain common uses of activity diagram.

Q.II Answer the following questions.

1. Explain which diagrams are called as an interaction diagram and also describe these diagrams are used to model which aspect of system.
2. Draw and explain sequence diagram with its components.
3. Explain the collaboration diagram with example.
4. Explain features that differentiate between collaboration diagram and sequence diagram.
5. Explain generalization, include and extend relationship among use case with example.
6. Which aspect of system can be modeled with use case? Describe uses of the use case diagram.
7. Explain the activity diagram with an example.
8. Explain common uses of interaction diagram.

9. Explain use diagram with an example.
10. What is a Component Diagram? How to create it?
11. With suitable example describe collaboration diagram.
12. Write difference between deployment and collaboration diagrams.
13. Compare deployment and component diagrams.
14. Compare component and collaboration diagrams.

Q.III Define the terms:

1. Interaction
2. Link
3. Use Case and Actor
4. State
5. Transitions
6. Branching
7. Forking and Joining
8. Swimlanes
9. Object flow
10. Component and Classes
11. Component and Interfaces.

Previous Exam Questions

Summer 2018

1. What is joining? [2 M]

Ans. Refer to Section 6.5.3.

2. What is lifeline? [2 M]

Ans. Refer to Section 6.4.1.3.

3. What is use cases? State include and extend relationship among use cases with sample. [4 M]

Ans. Refer to Section 6.2.5.

4. Define sequence diagram. Explain different kinds of its notations. [4 M]

Ans. Refer to Section 6.4.1.

5. Define term: Forking [1 M]

Ans. Refer to Section 6.5.3.

6. Railway reservation system is a system used for booking tickets over internet - Any customer can book tickets for different trains. Customer can book a ticket only if the tickets are available. Customer searches for the availability of ticket then if the tickets are available the books the ticket by initially filling details in a form. Tickets can be booked in two ways by i-ticket or by e-ticket booking.

   In case of i-ticket booking customer can book the ticket online and the tickets are couriered to particular customer at their address, but in case of e-ticket booking and canceling ticket are booked and cancelled online sitting at the home and customer himself has to take print of the ticket but in both the cases amount for tickets are deducted from customer's amount.

   For cancellation of ticket the customer's has to go at reservation office then fill cancellation form and ask the clerk to cancel the ticket then the refund is transferred to customer's account. After booking ticket the customer has to check out by paying fare amount to clerk.

Consider above situation. Draw the following UML diagrams:

(a) Use case diagrams [4 M]

Ans. Refer to Section 6.3.2.

(b) Activity diagrams [4 M]

Ans. Refer to Section 6.5.4.

(c) Sequence diagrams [4 M]

Ans. Refer to Section 6.4.1.4.

Winter 2018

1. Explain which diagrams are called as an Interaction diagram and explain these diagrams are used to model which aspect of system. [4 M]

Ans. Refer to Section 6.4.

2. When the customer arrives at the post check point with the items to purchase, the cashier records each item price and add the item information to the running sales transaction. The description and price of the current items are displayed. On completion of the item entry the cashier informs the sales totals and tax to the customer. The customer chooses payment type (cash, cheque, credit/debit). After the payment is made the system generates a receipt and automatically updates the inventory, the cashier handovers the receipt to the customer.

   Consider above situation, draw the following UML diagrams: [12 M]

(a) Use case diagram.

Ans. Refer to Section 6.3.4.

(b) Activity diagram.

Ans. Refer to Section 6.5.4.

(d) Sequence diagram.

Ans. Refer to Section 6.4.1.4.

Summer 2019

1. What is Swimlanes. [2 M]

Ans. Refer to Section 6.5.3.

2. What is Interaction diagram? Describe sequence diagram with example. [4 M]

Ans. Refer to Sections 6.4, 6.4.1.

3. Draw use case diagram for Hospital Management System. [4 M]

Ans. Refer to Section 6.3.2.

4. The case study titled Library Management System is Library management software for the purpose of monitoring and controlling the transactions in a library.

   The case study on the library management system gives us the complete information about the library and the daily transaction done in a library. We need to maintain the record of new and retrieve the details of books available in the library which mainly focuses on basic operations in a library like adding new members, new books and up new information, searching books and members and facility to borrow and return books.

If features a familiar and well thought out, an attractive user interface, combined with strong searching insertion and reporting capabilities the report generation facility of library system helps to get a good idea of which are the books borrowed by the members, makes users possible to generate hard copy.

Consider above situation, draw the following UML diagrams: [12 M]

(a) Sequence diagram.

Ans. Refer to Section 6.4.1.4.

(b) Use case diagram.

Ans. Refer to Section 6.3.2.

(c) Activity diagram.

Ans. Refer to Section 6.5.4.

■■■

7... Architectural Modeling

Learning Objectives ...

- To understand the design of the System Architecture.
- To understand the Component diagram and Deployment diagram.

7.1 COMPONENT

- A component in the Unified Modeling Language represents a modular part of a system that encapsulates the state and behavior of a number of classifiers. Its behavior is defined in terms of provided and required interfaces, is self-contained, and substitutable. A number of UML standard stereotypes exist that apply to components.
- Component is a physical part of an information system that exists at development time.

7.2 COMPONENT DIAGRAMS [S-19]

- Component diagram shows organizations and dependencies among a set of components.
- Component diagrams describe the organization of physical software components, including source code, run-time code and executables.
- The component diagram represents at a high level, what components form part of the system and how they are interrelated.

7.2.1 Concept

- Component Based Architecture (CBD) is the design of a software system made up of multiple components.
- Component diagrams are a special kind of UML diagram to describe a static implementation view of a system. Component diagrams consist of physical components like libraries, files, folders etc.
- Component diagram shows components, provided and required interfaces, ports, and relationships between them.
- Use component diagrams when you are dividing your system into components and want to show their interrelationships through interfaces or the breakdown of components into a lower-level structure.

7... Architectural Modeling

Learning Objectives ...

- To understand the design of the System Architecture.
- To understand the Component diagram and Deployment diagram.

7.1 COMPONENT

- A component in the Unified Modeling Language represents a modular part of a system that encapsulates the state and behavior of a number of classifiers. Its behavior is defined in terms of provided and required interfaces, is self-contained, and substitutable. A number of UML standard stereotypes exist that apply to components.
- Component is a physical part of an information system that exists at development time.

7.2 COMPONENT DIAGRAMS [S-19]

- Component diagram shows organizations and dependencies among a set of components.
- Component diagrams describe the organization of physical software components, including source code, run-time code and executables.
- The component diagram represents at a high level, what components form part of the system and how they are interrelated.

7.2.1 Concept

- Component Based Architecture (CBD) is the design of a software system made up of multiple components.
- Component diagrams are a special kind of UML diagram to describe a static implementation view of a system. Component diagrams consist of physical components like libraries, files, folders etc.
- Component diagram shows components, provided and required interfaces, ports, and relationships between them.
- Use component diagrams when you are dividing your system into components and want to show their interrelationships through interfaces or the breakdown of components into a lower-level structure.

Purpose of the Component Diagram:

1. Visualize the components of a system.
2. Construct executables by using forward and reverse engineering.
3. Describe the organization and relationships of the components.

- A component is a logical, replaceable part of a system that conforms to and provides the realization of a set of interfaces.
- Good components define hard abstractions with well-defined interfaces, making it possible to easily replace older components with newer, compatible ones.
- Interfaces bridge your logical and design models. For example, you may specify an interface for a class in a logical model, and that same interface will carry over to some design component that realizes it.
- Interfaces allow you to build the implementation of a component using smaller components by wiring ports on the components together.
- Component diagrams illustrate the pieces of software, embedded controllers, etc., that will make up a system.
- A component diagram has a higher level of abstraction than a class diagram; usually a component is implemented by one or more classes (or objects) at run-time. They are building blocks so a component can eventually encompass a large portion of a system.
- Fig. 7.1 demonstrates some components and their inter-relationships. Assembly connectors "link" the provided interfaces supplied by "Product" and "Customer" to the required interfaces specified by "Order". A dependency relationship maps a customer's associated account details to the required interface; "Payment", indicated by "Order".

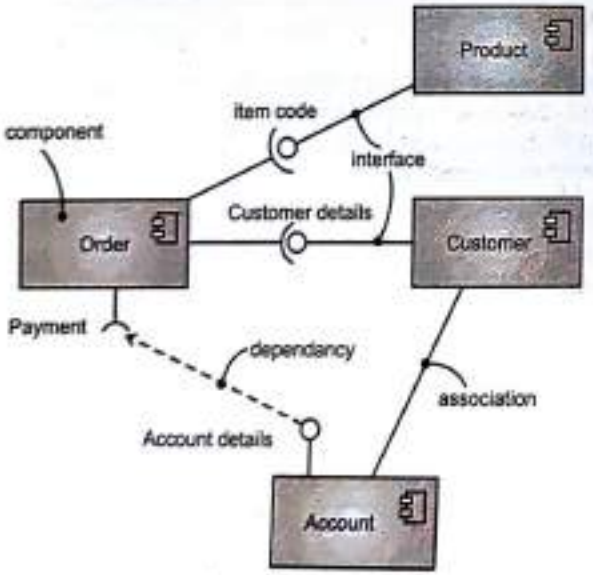
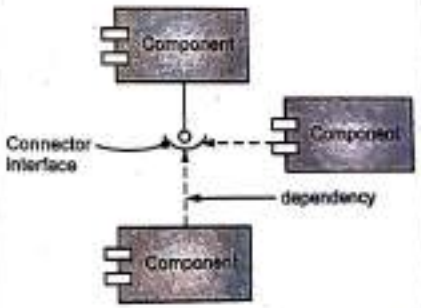


Fig. 7.1: Component diagram

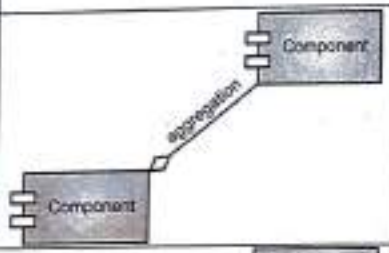
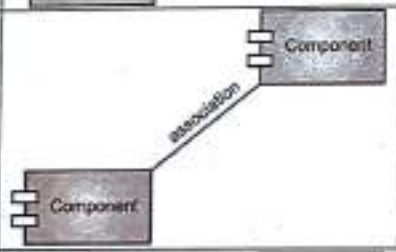
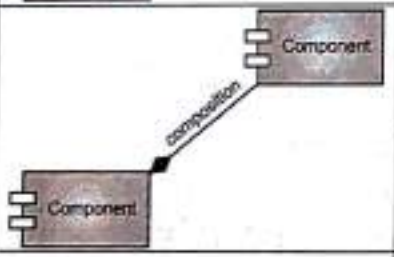
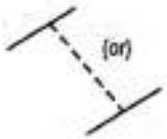
7.2.2 Notations

- Following table shows various notations of Components diagram:

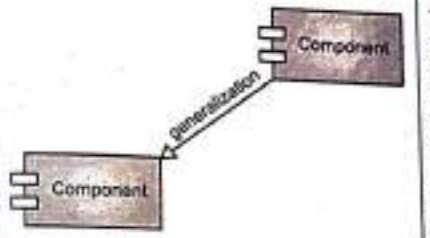
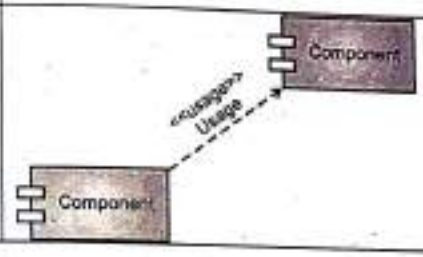
Table 7.1: Notations of Components Diagram

Name	Symbol	Description
1. Component		A component is a physical building block of the system. It is represented as a rectangle with tabs. A component represents a modular part of a system that encapsulates its contents and whose manifestation is replace-able within its environment.
2. Interface		An interface describes a group of operations used or created by components. An interface is a kind of classifier that represents a declaration of a set of coherent public features and obligations.
3. Dependencies		Draw dependencies among components using dashed arrows. A dependency is a relationship that signifies a single or a set of model elements that requires other model elements for their specification or implementation.

contd. ...

4. Port		A port is a property of a classifier that specifies a distinct interaction point between that classifier and its environment or between the (behavior of the) classifier and its internal parts.
5. Note		A note (comment) gives the ability to attach various remarks to elements.
6. Aggregation		A kind of association that has one of its end marks shared as a kind of aggregation, meaning that it has a shared aggregation.
7. Association (Link)		An association specifies a semantic relationship (link) that can occur between typed instances.
8. Composition		An association may represent a composite aggregation (i.e., a whole/part relationship).
9. Constraint		A condition or restriction expressed in natural language text or in a machine readable language for the purpose of declaring some of the semantics of an element.

contd. ...

10. Generalization		A generalization is a taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the more general classifier.
11. Usage		Usage is a relationship in which one element requires another element (or set of elements) for its full implementation or operation.

7.2.3 Examples

[W-18, S-19]

1. Component Diagram for Library Management System:

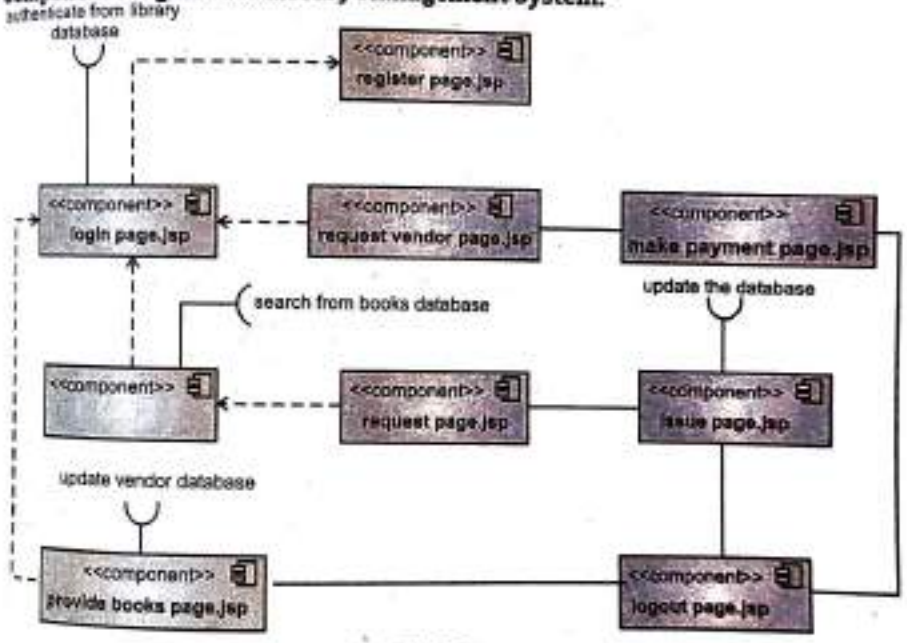


Fig. 7.2

2. Component Diagram for Online Shopping:

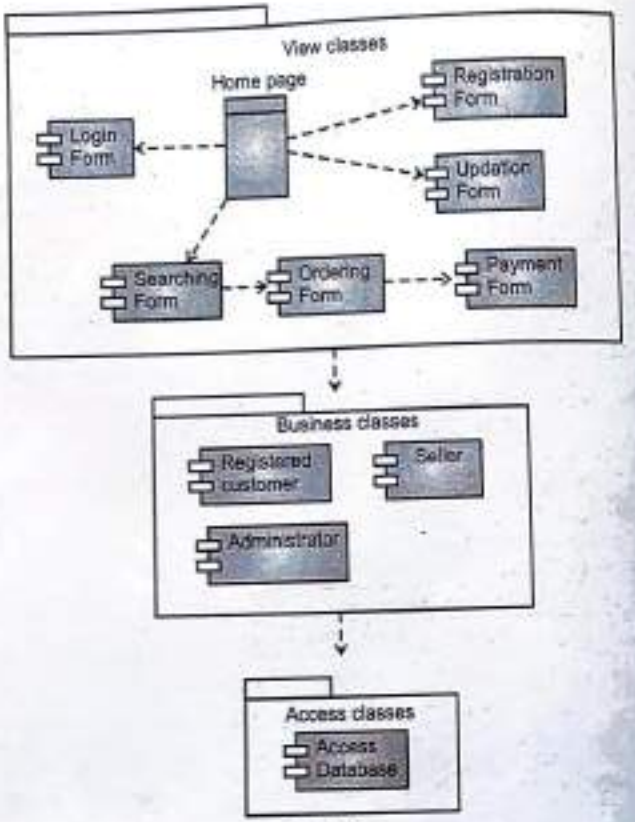


Fig. 7.3

3. Component Diagram for Railway Ticket Reservation System:

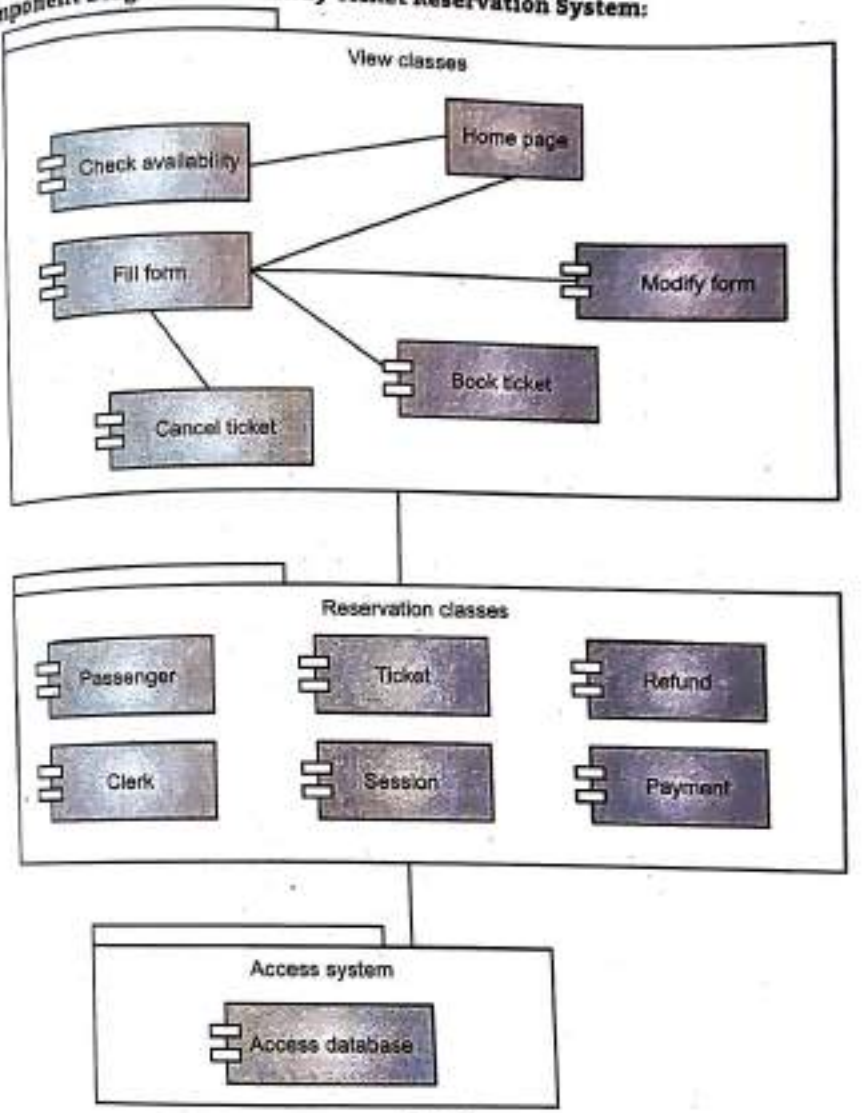


Fig. 7.4

4. **Component Diagram for ATM Transaction System:**

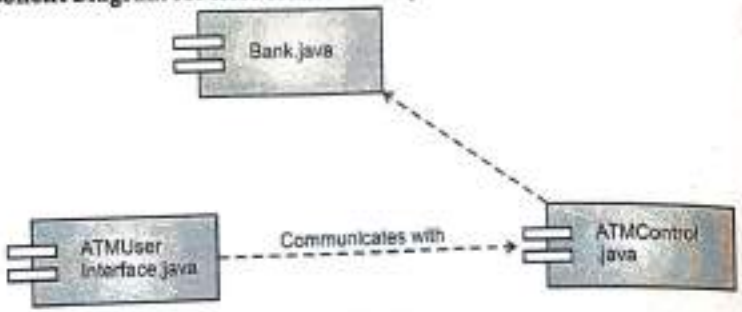


Fig. 7.5

7.3 DEPLOYMENT DIAGRAMS [W-18]

- The deployment diagrams show the physical layout of the network and where the various components will reside.
- The deployment diagram shows how a system will be physically deployed in the hardware environment. Its purpose is to show where the different components of the system will physically run and how they will communicate with each other.
- Since, the diagram models the physical runtime; a system's production staff will make considerable use of this diagram.

7.3.1 Concept

- A deployment diagram is a diagram that shows the configuration of run time processing nodes and the artifacts that live on them.
- Graphically, a deployment diagram is a collection of vertices and arcs.

Purpose of Deployment Diagrams:

1. Visualize hardware topology of a system.
2. Describe the hardware components used to deploy software components.
3. Describe runtime processing nodes.

- Deployment diagrams commonly contain nodes, dependency, association relationships etc.
- Like all other diagrams, deployment diagrams may contain notes and constraints. Deployment diagrams may also contain artifacts, each of which must live on some node.
- With the UML, you use deployment diagrams to visualize the static aspect of these physical nodes and their relationships and to specify their details for construction, as shown in Fig. 7.6.

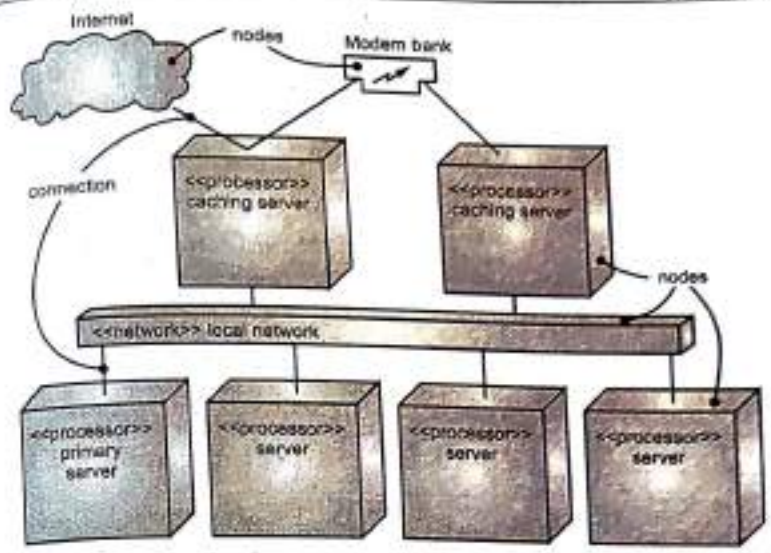


Fig. 7.6: Deployment diagram

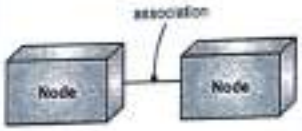
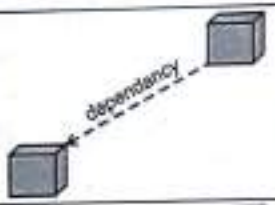
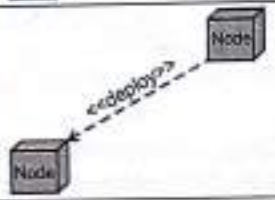
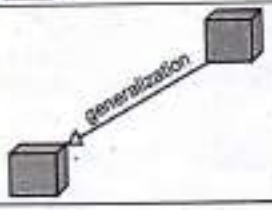
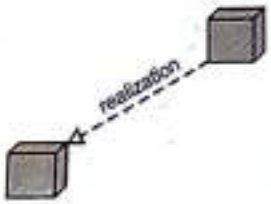
7.3.2 Notations

- Following table gives various notations in deployment diagram.

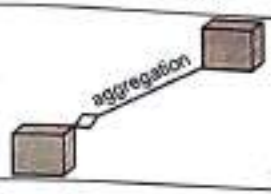
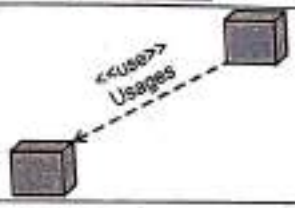
Table 7.2: Notations of Deployment Diagram

Name	Symbol	Description
1. Node	Node	A node is a physical resource that executes code components. A node is a computational resource upon which artifacts may be deployed for execution.
2. Device node	<<device>> Node	A device node is a physical computational resource with processing capability upon which artifacts may be deployed for execution. Devices may be complex (i.e., they may consist of other devices).
3. Execution Environment Node	<<executionEnvironment>> Node	An execution environment node is a node that offers an execution environment for specific types of components that are deployed on it in the form of executable artifacts.

contd. ...

4. Association		It refers to a physical connection i.e., link between nodes. It specifies a semantic relationship that can occur between typed instances.
5. Component		A component defines its behavior in terms of provided and required interfaces.
6. Dependency		It is a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation.
7. Deployment		A deployment is the allocation of an artifact or artifact instance to a deployment target.
8. Generalization		It is a taxonomic relationship between a more general classifier and a more specific classifier.
9. Port		A Port may specify the services a classifier provides (offers) to its environment as well as the services that a classifier expects (requires) of its environment.
10. Realization		Realization is a specialized abstraction relationship between two sets of model elements, one representing a specification (the supplier) and the other represents an implementation of the latter (the client).

contd. ...

11. Aggregation		It is a kind of association that has one of its end marks shared as a kind of aggregation, meaning that it has a shared aggregation.
12. Artifact		An artifact is the specification of a physical piece of information that is used or produced by a software development process, or by deployment and operation of a system. Examples of artifacts include model files, source files, scripts, and binary executable files, a table in a database system, a development deliverable, or a word-processing document, a mail message.
13. Interface		It is a kind of classifier that represents a declaration of a set of coherent public features and obligations. It specifies a contract; any instance of a classifier that realizes the interface must fulfill that contract.
14. Note		It gives the ability to attach various remarks (comments) to elements.
15. Usage		It is a relationship in which one element requires another element or set of elements for its full implementation or operation.

7.3.4 Common terms in Deployment Diagrams

1. **Nodes and Node Names:**

- Nodes are an important building block in modeling the physical aspects of a system.
- A node is a physical element that exists at run time and represents a computational resource, generally having at least some memory and, often, processing capability.
- We use nodes to model the topology of the hardware on which our system executes. A node typically represents a processor or a device on which artifacts may be deployed. Good nodes crisply represent the vocabulary of the hardware in your solution domain.

- When we design a software-intensive system, we have to consider both its logical and physical dimensions.
  (i) On the logical side, you will find classes, interfaces, collaborations, interactions, and state machines.
  (ii) On the physical side, you will find artifacts (which represent the physical packaging of these logical things) and nodes (which represent the hardware on which these artifacts are deployed and executed).
- The UML provides a graphical representation of node as shown in Fig. 7.7. A node is a physical element that exists at run time and represents a computational resource, generally having at least some memory and, often, processing capability. Graphically, a node is rendered as a cube.
- Every node must have a name that distinguishes it from other nodes. Name alone is known as a simple name; a qualified name is the node name prefixed by the name of the package in which that node lives. A node is typically drawn showing only its name.
- Fig. 7.7 shows nodes with simple and qualified names.

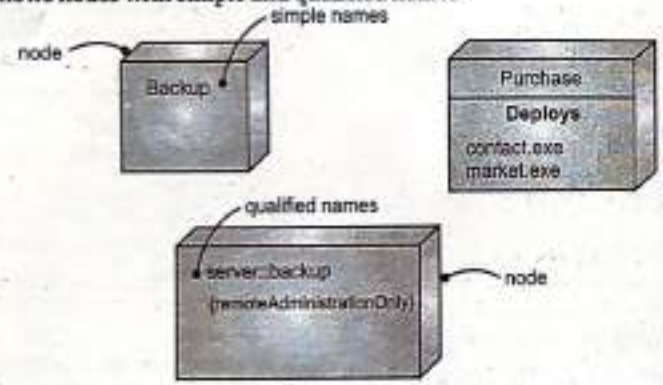


Fig. 7.7: Nodes with simple and qualified names

2. Node Instances:

- A node instance can be shown on a diagram. An instance can be distinguished from a node by the fact that its name is underlined and has a colon before its base node type.
- An instance may or may not have a name before the colon.
- Fig. 7.8 shows a named node instance of a computer.

Fig. 7.8: Node Instances

3. Nodes and Components:

- In many ways, nodes are a lot like components i.e. both nodes and components have names. Both nodes and components may participate in dependency, generalization and association relationships. Both nodes and components may be nested.
- Both nodes and components may have instances and may be participants in interactions.
- However, there are some significant differences between Nodes and Components.
  (i) Components are things that participate in the execution of a system, while nodes are things that execute components.
  (ii) Components represent the physical packaging of otherwise logical elements, while modes represent the physical deployment of components.
- Fig. 7.9 shows, the relationship between a node and components it deploys can be shown explicitly by using a dependency relationship.

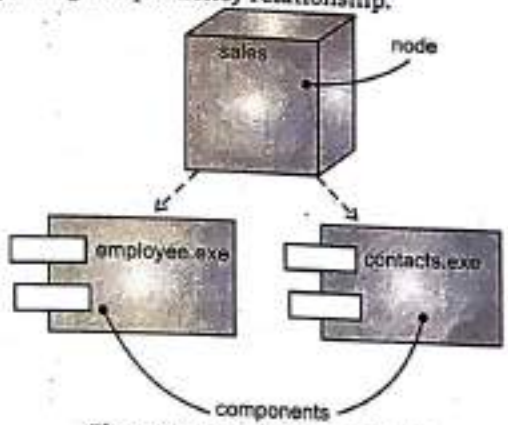


Fig. 7.9: A node and components

- A set of objects or components that are allocated to a node as a group is called a 'distribution unit'.

4. Node Stereotypes:

- A number of standard stereotypes are provided for nodes, namely «cdrom», «cd-rom», «computer», «disk array», «pc», «pc client», «pc server», «secure», «server», «storage», «unix server», «user pc».
- These will display an appropriate icon in the top right corner of the node symbol.

Fig. 7.10: Node Stereotypes

5. Connections (Relationship):

Association:

- The most common kind of relationship among nodes is an Association.

- Association refers to a physical connection or link between the nodes.
- A communication association between nodes indicates a communication path between the nodes that allows components on the nodes to communicate with one another.
- A communication association is shown as a solid-line between nodes. Fig. 7.11 shows that the Business-Processing Server has a communication association with the Desktop Client, Printer, and Database Server nodes.

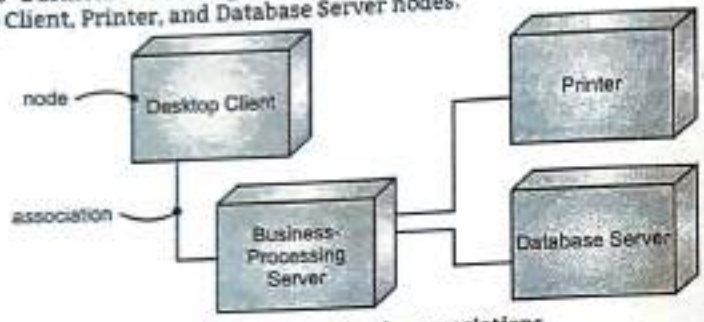


Fig. 7.11: Communication associations

(ii) **Dependency:**

- A dependency is a relationship that indicates that a model element is in some way dependent on another model element.
- All model elements must exist on the same level of abstraction or realization. A dependency is a semantic relationship between two things in which a change to one thing (the independent thing) may affect the semantics of the other thing (the dependent thing).
- The dependency of a node on components is depicted using a dashed line, (See Fig. 7.12). This indicates that a node uses the services of the components that are executing on another node.
- For example, you can depict the dependency relationship between an Application Server node and a database server component, as shown in the Fig. 7.12.

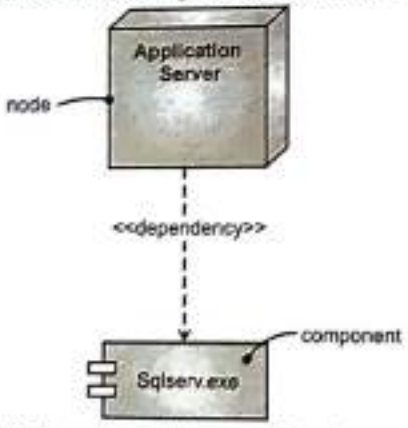


Fig. 7.12: Dependency between node and component

(iii) **Generalization:**

- A generalization is a relationship between a more general element and a more specific element.
- Generalization is a specialization/generalization relationship in which objects of the specialized element (the child) are substitutable for objects of the generalized element (the parent).

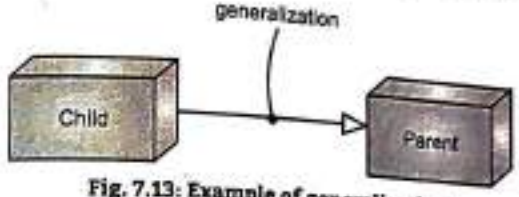


Fig. 7.13: Example of generalization

(iv) **Realization:**

- Realization is a relationship between interfaces and classes or components that realize them.
- Realization defines a relationship in which one class specifies something that another class will perform.
- Example: The relationship between an interface and the class that realizes or executes that interface.

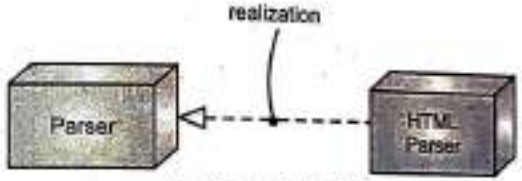


Fig. 7.14: Realization

6. **Nodes and Artifacts:**

- In many ways, nodes are a lot like artifacts: both have names; both may participate in dependency, generalization, and association relationships; both may be nested; both may have instances; both may be participants in interactions. However, there are some significant differences between nodes and artifacts.
- Artifacts are things that participate in the execution of a system; nodes are things that execute artifacts.
- Artifacts represent the physical packaging of otherwise logical elements; nodes represent the physical deployment of artifacts.
- This first difference is the most important. Simply put, nodes execute artifacts; artifacts are things that are executed by nodes.
- The second difference suggests a relationship among classes, artifacts, and nodes. In particular, an artifact is the demonstration of a set of logical elements, such as classes and collaborations, and a node is the location upon which artifacts are deployed.
- A class may be marked by one or more artifacts, and, in turn, an artifact may be deployed on one or more nodes.

- Fig. 7.15 shows, the relationship between a node and the artifacts it deploys can be shown explicitly by using nesting. Most of the time, you won't need to visualize these relationships graphically but will indicate them as a part of the node's specification, for example, using a table.

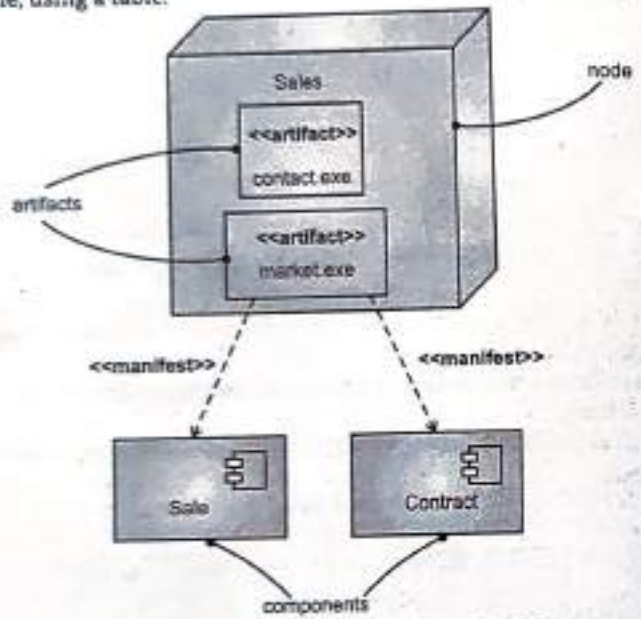


Fig. 7.15: Relationship between Nodes and Artifacts

7.3.5 Examples

1. Deployment Diagrams for Online Hospital Management System:

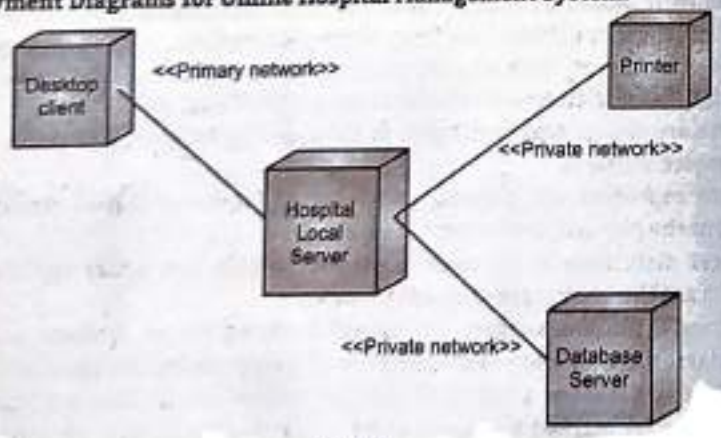


Fig. 7.16

2. Deployment Diagram for ATM Machine:

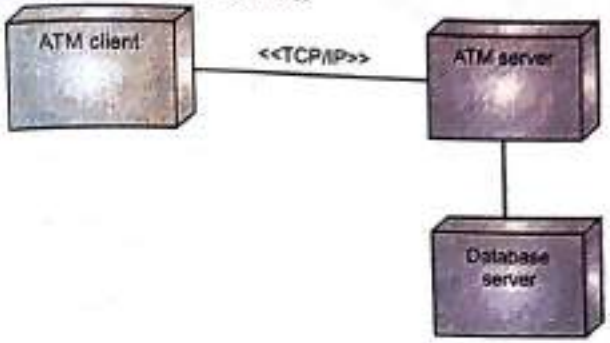


Fig. 7.17

3. Deployment Diagram for Online Shopping:

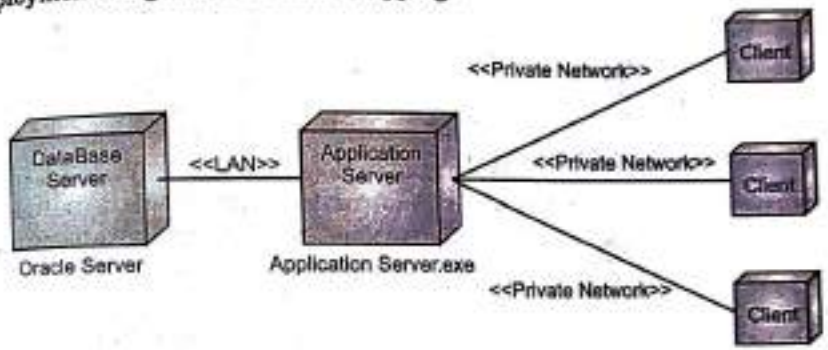


Fig. 7.18

4. Deployment Diagram for Railway Ticket Reservation System:

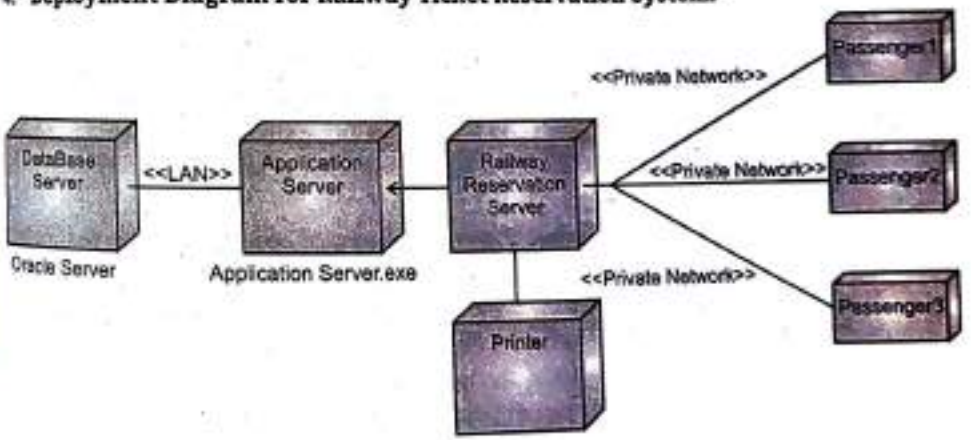


Fig. 7.19

5. **Deployment Diagram of an Order Management System:**

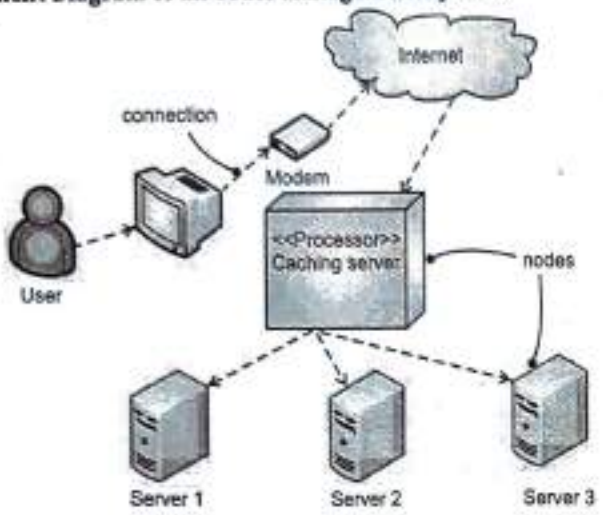


Fig. 7.20

7.4 COLLABORATION DIAGRAM [S-19]

- A collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML).

Purpose:

- A collaboration diagram shows the objects and relationships involved in an interaction, and the sequence of messages exchanged among the objects during the interaction.

7.4.1 Concept

- Collaboration diagrams are also interaction diagrams. They convey the same information as sequence diagrams, but they focus on object roles instead of the times that messages are sent.
- In a sequence diagram, object roles are the vertices and messages are the connecting links.
- A collaboration diagram emphasizes the organization of objects that are part of the diagram.
- A collaboration diagram is formed by placing the objects that participate in interaction as vertices in a graph and the links that connect objects as arcs of graph.
- These links can be shown with messages that objects send and receive. This gives the reader a clear view to flow of control in structural organization of objects that collaborate.

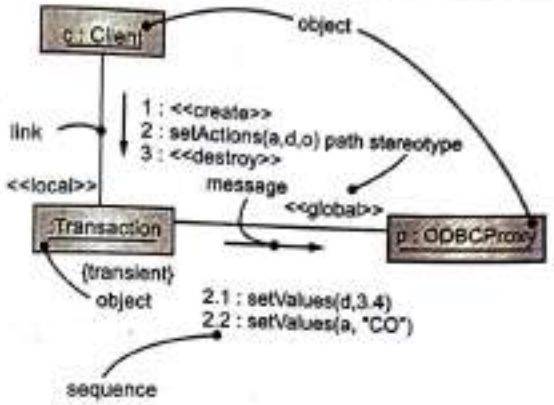


Fig. 7.21: Collaboration diagram

- First, there is the **path**. To indicate how one object is linked to another, we can attach a path stereotype to the far end of a link (such as `<<local>>`, indicating that the designated object is local to the sender). Typically, we will only need to reduce the path of the link explicitly for local, parameter, `<<global>>` and self (but not association) paths.
- Second, there is the **sequence number**. To indicate the time order of a message, we prefix the message with a number (starting with the message numbered 1), increasing monotonically for each new message in the flow of control (2, 3, and so on). To show nesting, we use Dewey decimal numbering (1 is the first message; 1.1 is the first message nested in message 1; 1.2 is the second message nested in message 1; and so on). Along the same link, we can show many messages (possibly being sent from different directions) and each will have a unique sequence number.

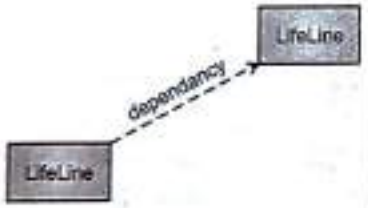
7.4.2 Notations [S-18]

- Following table gives various notations used in collaboration diagram.

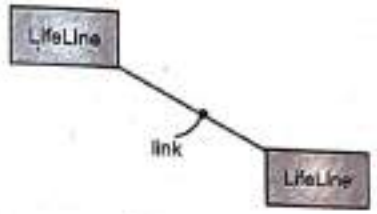
Table 7.3: Notations of Collaboration Diagram

Name	Notation	Description
1. Actor	 Actor	An actor in a collaboration diagram represents the person, software, hardware, or other agent external to the system that is interacting with the system.

contd. ...

2. Instance (object)		An instance specification is a model element that represents an instance in a modeled system.
3. Active object		An active object is an instance that owns a thread of control and can initiate control activity. For example, processes and tasks are active objects. Active objects can contain other symbols and links among them.
4. Lifeline		It represents individual participants in the interaction.
5. Message		A message defines a particular communication between Lifelines of an Interaction. Call message is a kind of message that represents an invocation of operation of the target lifeline.
6. Dependency		A dependency is a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation. This means that the complete semantics of the depending elements is either semantically or structurally dependent on the definition of the supplier element(s).
7. Generalization		A generalization is a taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the more general classifier.

contd. ...

8. Link		An association declares that there can be links between instances of the associated types. A link is a tuple with one value for each end of the association, where each value is an instance of the type of the end.
9. Note		A note (comment) gives the ability to attach various remarks to elements. A comment carries no semantic force, but may contain information that is useful to a modeler.

7.4.3 Examples

1. **Collaboration diagram for ATM:** [S-19]

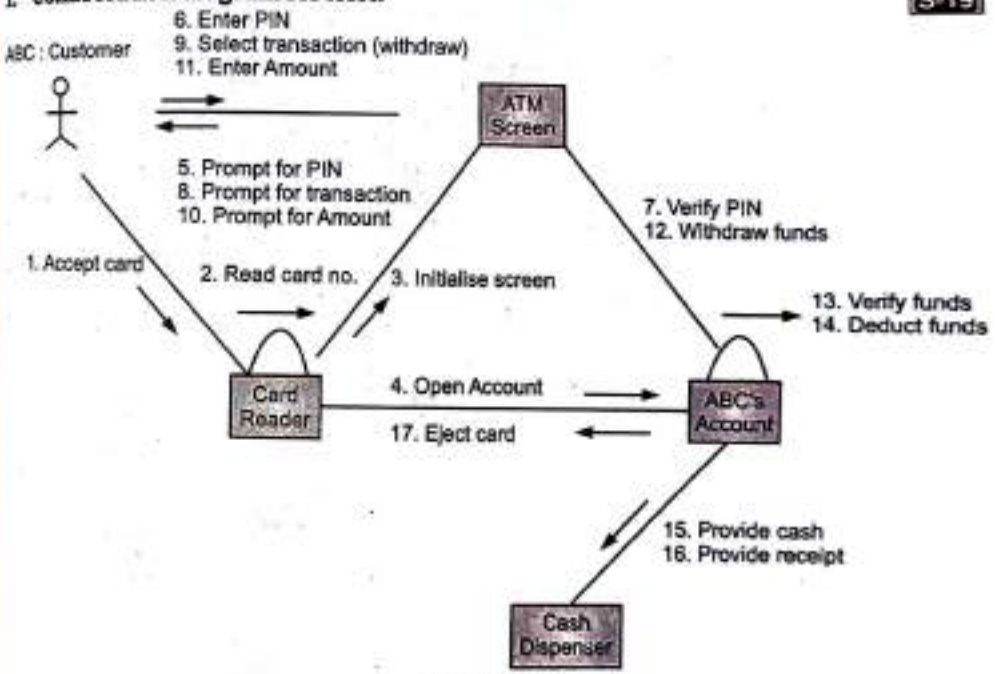


Fig. 7.22

2. Collaboration Diagram for Railway Reservation System:

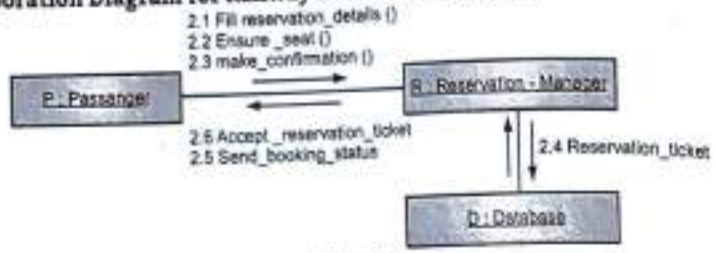


Fig. 7.23

3. Collaboration Diagram of "Make Phone Call":

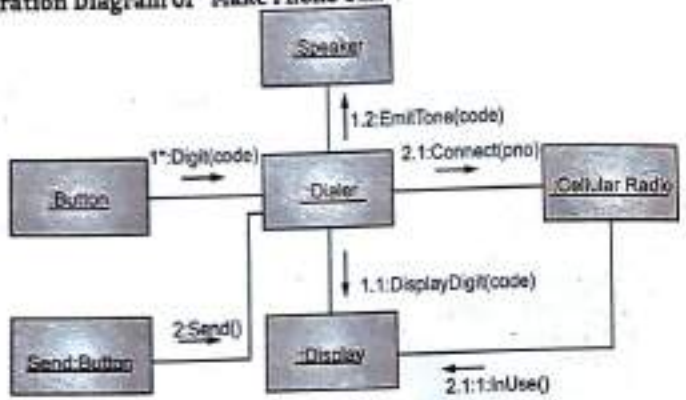


Fig. 7.24

4. Library Management System

(i) Collaboration diagram for issuing Book:

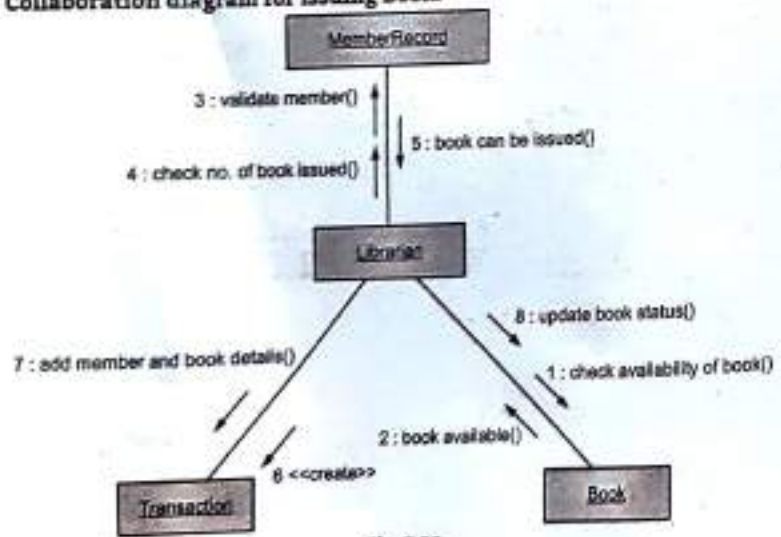


Fig. 7.25

(ii) Collaboration Diagram for Returning Book in Library:

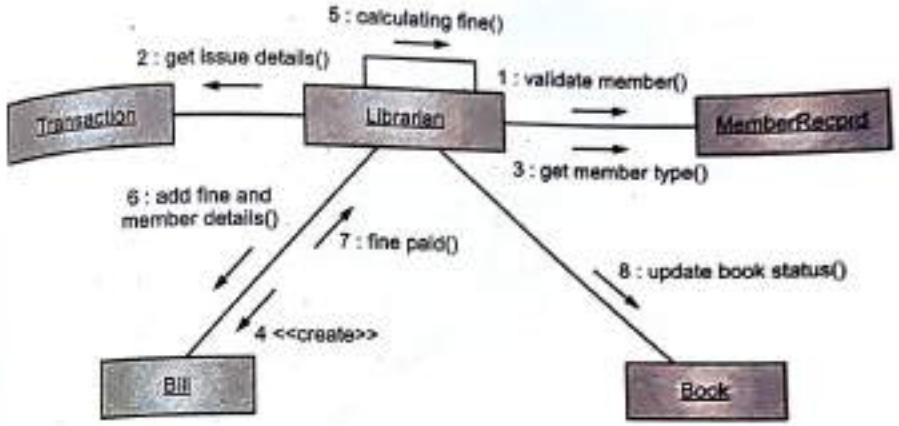


Fig. 7.26

5. Collaboration of Diagram for Hospital Management System:

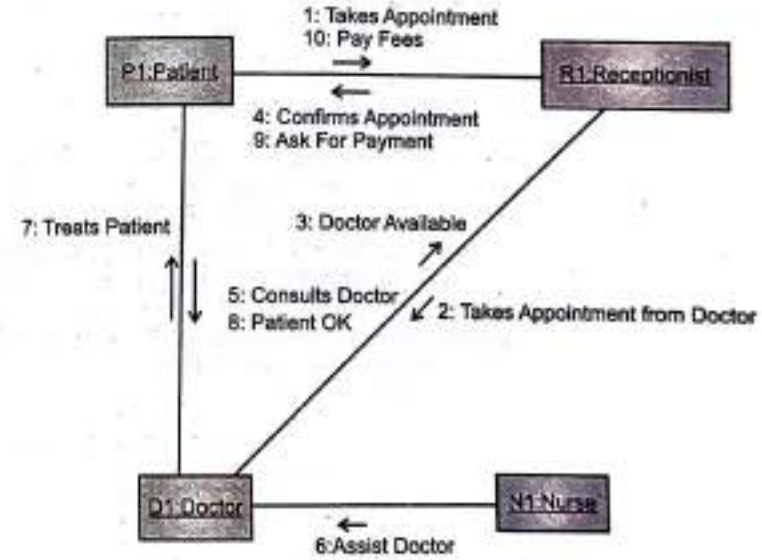


Fig. 7.27

6. Collaboration Diagram for Course Registration System:

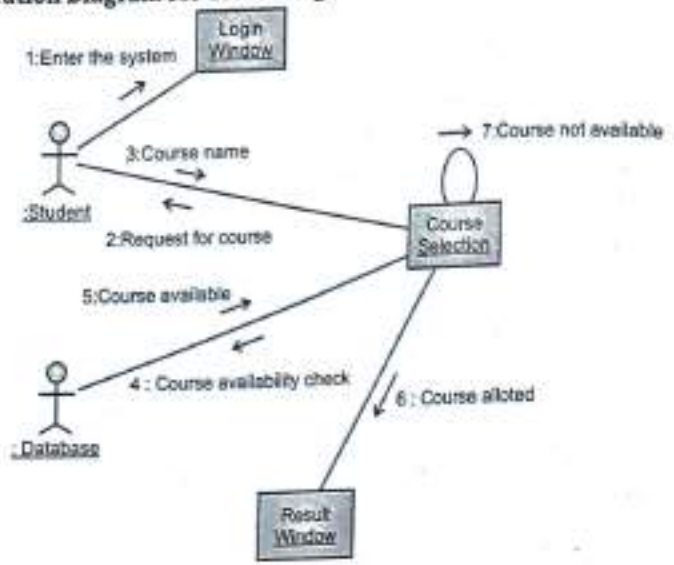


Fig. 7.28

Summary

- Component is a physical part of an information system that exists at development time.
- Component diagrams describe the organization of physical software components, including source code, run-time code and executables.
- The deployment diagram shows how a system will be physically deployed in the hardware environment. Its purpose is to show where the different components of the system will physically run and how they will communicate with each other.
- Graphically, a deployment diagram is a collection of vertices and arcs.
- A node is a physical element that exists at run time and represents a computational resource, generally having at least some memory and, often, processing capability.
- A collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML).

Check Your Understanding

1. Which among the following are not the valid notations for package and component diagrams?
   (a) Notes (b) Box
   (c) Extension Mechanisms (d) Packages
2. What types of units does component follow?
   (a) Modular Unit (b) Replaceable Unit
   (c) Unit with well-defined interface (d) All of the mentioned

3. Components can be represented by which of the following?
   (a) Component symbols (b) Stereotypes
   (c) Rectangular boxes (d) Component symbols & Stereotypes
4. What does a component diagram consist of?
   (a) Components, their Relationship to the environment
   (b) Packages and dependency
   (c) Internal structure
   (d) Internal structure, Components & their Relationship to the environment
5. Which among these are the common notations for deployment diagrams?
   (a) Artifacts and nodes (b) Stereotypes
   (c) Components (d) All of the mentioned
6. Which of the following statements is true?
   (a) A UML artifact is any physical representation of data used or produced during software development or software product operation.
   (b) A node is a computational resource.
   (c) All of the mentioned
   (d) None of the mentioned
7. Which of these are types of nodes used in the deployment diagram?
   (a) Device (b) Execution Environment
   (c) Artifact (d) Device & Execution Environment
8. Which are the ways to represent nodes in a deployment diagram?
   (a) Nodes instances are underlined identifiers of the form name : type.
   (b) The name may be left off, indicating an unnamed instance of the type.
   (c) The type may be left off, indicating a named instance with an unspecified type.
   (d) All of the mentioned
9. What does a deployment diagram consist of?
   (a) Computational resource
   (b) Communication path between resources
   (c) Artifacts that execute resource
   (d) All of the mentioned
10. Which of the following is incorrect in the deployment diagram?
    (a) Communication connections between nodes are shown by communication paths.
    (b) Communication paths are represented by dotted lines.
    (c) Artifacts are deployed inside nodes where they reside and execute.
    (d) None of the mentioned

Answers

1. (b)	2. (d)	3. (d)	4. (d)	5. (a)	6. (c)	7. (d)	8. (d)	9. (d)	10. (b)

Practice Questions

Q.I Answer the following questions in short.

1. What is a node?
2. What are Node instances?
3. Enlist various types of components.
4. Define the deployment diagram.
5. Write the difference between Node and components?

Q.II Answer the following questions.

1. Explain the term Component with an example.
2. Explain use of component diagram.
3. Define the collaboration diagram and describe the various elements of it with example.
4. Explain the components of the Deployment diagram.
5. Compare component and collaboration diagrams.
6. Compare deployment and component diagrams.
7. Draw a collaboration diagram for ATM system.

Q.III Define the terms.

1. Link
2. Lifeline
3. Actor
4. Device Node

Previous Exam Questions

Summer 2018

1. What is lifeline? **[2 M]**

Ans. Refer to Section 7.4.2.2.

Winter 2018

1. Draw component diagram for online shopping. **[4 M]**

Ans. Refer to Sections 7.2 and 7.2.3.

2. What is Deployment diagram? State any four notations of deployment diagram. **[4 M]**

Ans. Refer to Section 7.3.

3. Construct a design element for point of the sale terminal management system that can be used for buying and selling of goods in the retail shop. When the customer arrives at the post check point with the items to purchase, the cashier records each item price and add the item information to the running sales transaction. The description and price of the current items are displayed. On completion of the item entry the cashier informs the sales totals and tax to the customer. The customer chooses payment type (cash, cheque, credit/debit). After the payment is made the system generates a receipt and automatically updates the inventory, the cashier handovers the receipt to the customer.
Consider above situation, draw the collaboration diagram. **[4 M]**

Ans. Refer to Section 7.4.2.3.

Summer 2019

1. Define the collaboration diagram. Draw collaboration diagram for ATM. **[4 M]**

Ans. Refer to Sections 7.4 and 7.4.2.3.

■■■

8... Object Oriented Analysis

Learning Objectives ...

- To understand the Object-oriented analysis in detail
- To understand the iterative development and rational unified process

8.1 INTRODUCTION

- Analysis emphasizes an investigation of the problem and requirements, rather than a solution. In Object Oriented Analysis (OOA), there is an emphasis on finding and describing the objects or concepts in the problem domain.
- The use of modeling to define and analyze the requirements necessary for success of a system.
- Object-oriented analysis is a process that groups items that interact with one another, typically by class, data or behavior, to create a model that accurately represents the intended purpose of the system as a whole.
- The purpose of any analysis activity in the software life-cycle is to create a model of the system's functional requirements that is independent of implementation constraints.
- The main difference between object-oriented analysis and other forms of analysis is that by the object-oriented approach we organize requirements around objects, which integrate both behaviors (processes) and states (data) modeled after real world objects that the system interacts with.
- Object-oriented analysis examines system requirements from the perspective of classes and objects found in the specified domain. It focuses on what the system is supposed to do rather than how it will be done and looks at the behavior of the system independent of its domain.
- Object-oriented analysis looks at the real-world environment in which a system will operate, with this environment consisting of people and things interacting to create some result.
- The primary tasks in object-oriented analysis (OOA) are:
  1. Find the objects.
  2. Organize the objects.
  3. Describe how the objects interact.

8... Object Oriented Analysis

Learning Objectives ...

- To understand the Object-oriented analysis in detail
- To understand the iterative development and rational unified process

8.1 INTRODUCTION

- Analysis emphasizes an investigation of the problem and requirements, rather than a solution. In Object Oriented Analysis (OOA), there is an emphasis on finding and describing the objects or concepts in the problem domain.
- The use of modeling to define and analyze the requirements necessary for success of a system.
- Object-oriented analysis is a process that groups items that interact with one another, typically by class, data or behavior, to create a model that accurately represents the intended purpose of the system as a whole.
- The purpose of any analysis activity in the software life-cycle is to create a model of the system's functional requirements that is independent of implementation constraints.
- The main difference between object-oriented analysis and other forms of analysis is that by the object-oriented approach we organize requirements around objects, which integrate both behaviors (processes) and states (data) modeled after real world objects that the system interacts with.
- Object-oriented analysis examines system requirements from the perspective of classes and objects found in the specified domain. It focuses on what the system is supposed to do rather than how it will be done and looks at the behavior of the system independent of its domain.
- Object-oriented analysis looks at the real-world environment in which a system will operate, with this environment consisting of people and things interacting to create some result.
- The primary tasks in object-oriented analysis (OOA) are:
  1. Find the objects.
  2. Organize the objects.
  3. Describe how the objects interact.

4. Define the behavior of the objects.
5. Define the internals of the objects.

- Common models used in OOA are Use cases and Object models.
  1. **Use cases** describe scenarios for standard domain functions that the system must accomplish.
  2. **Object models** describe the names, class relations, (e.g. Circle is a subclass of Shape), operations, and properties of the main objects.
- Object–Oriented Analysis (OOA) is the procedure of identifying software engineering requirements and developing software specifications in terms of a software system's object model, which comprises interacting objects.
- Grady Booch has defined OOA as "Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain".

8.2 ITERATIVE DEVELOPMENT [S-19]

- Iterative process is a process for arriving at a decision or a desired result by repeating rounds of analysis or a cycle of operations.
- Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented.
- At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).
- In iterative development, a process is broken into a series of steps or iterations. Iteration is the act of repeating a process with the aim of approaching a desired goal, target or result.
- An iterative approach is shown in Fig. 8.1.

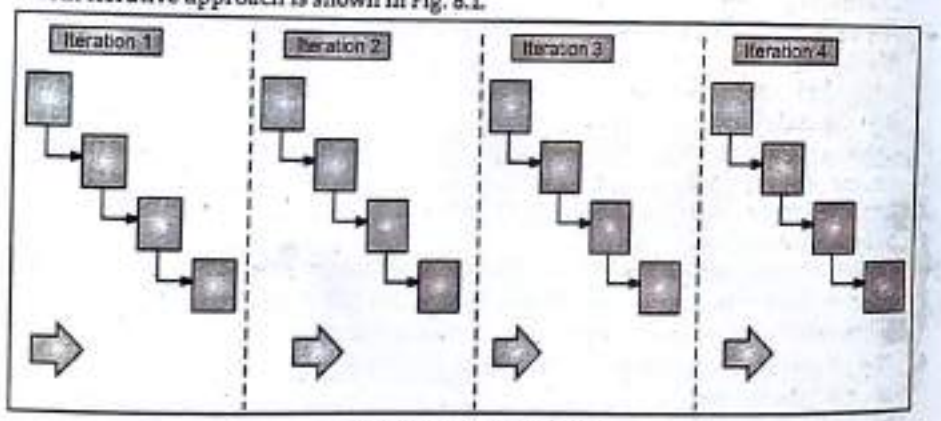


Fig. 8.1: Iterative Approach

- Iterative development lies at the heart of how OOA (Object Oriented Analysis) is best practiced.

Iterative and Evolutionary Development:

- Evolutionary development is an iterative and incremental approach to software development.
- Iterative and Incremental development is any combination of both iterative design or iterative method and incremental build model for software development.
- **Iterative and evolutionary development concepts:**
  1. **Iterative:** In this style, the development team is doing lots of good stuff like: stand-up meetings, planning meetings, short iterations, test driven development, code review, refactoring.
  2. **Incremental:** This style is mostly the same as Iterative; it looks similar to start with. The team is still (hopefully) doing good stuff and iterating.
  3. **Evolutionary:** Here, again the development team is iterating much as before. However, this time there is no requirements document. Work has begun with just an idea.
- Fig. 8.2 shows iterative and evolutionary development.

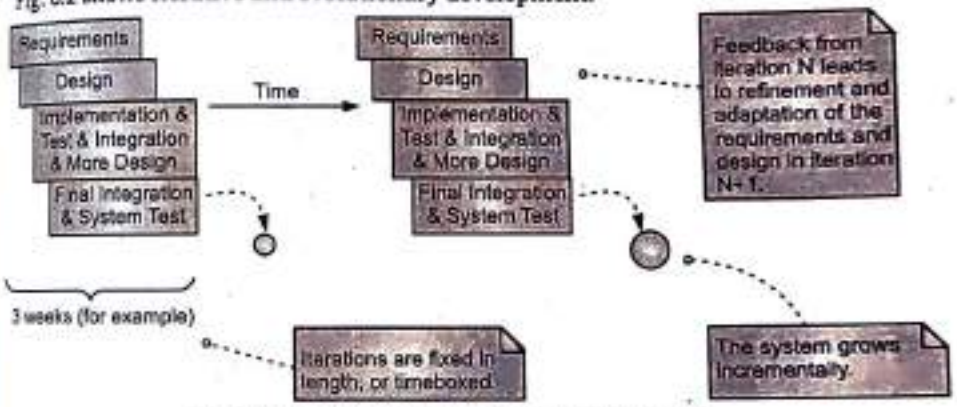


Fig. 8.2: Iterative and Evolutionary Development

- Contrasted with a sequential or "waterfall" lifecycle which involves early programming and testing of a partial system in repeating cycles.
- This model is named "waterfall model" because its diagrammatic representation resembles a cascade of waterfalls.
- Iterative and evolutionary development also normally assumes development starts before all the requirements are defined in detail; feedback is used to clarify and improve the evolving specifications.
- To contrast, waterfall values promoted big upfront speculative requirements and design steps before programming.
- Consistently, success/failure studies show that the waterfall is strongly associated with the highest failure rates for software projects and it was historically promoted due to belief or hearsay rather than statistically significant evidence.

- Iterative methods are associated with higher success and productivity rates and lower defect levels.
- Fig. 8.3 shows the waterfall approach shows software stages in a rigid linear sequence with no backtracking.

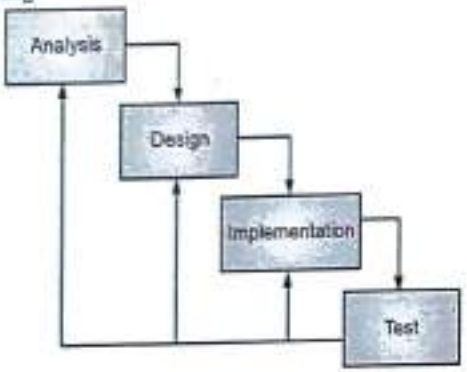


Fig. 8.3: Waterfall Approach

8.2.1 Phases of Iterative Development

- Iterative development is an approach to building software (or anything) in which the overall lifecycle is composed of several iterations in sequence.
- Each iteration is a self-contained mini-project composed of activities such as requirements analysis, design, programming, and test.
- The goal for the end of an iteration is an iteration release, a stable, integrated and tested partially complete system.
- Iterative development contains four major phases:
  1. Inception
  2. Elaboration
  3. Construction
  4. Transition.

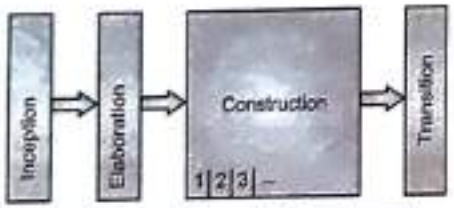


Fig. 8.4: Phase of Iterative Development

- Fig. 8.4 shows following phase of iterative development:

Phase 1 : **Inception:** Inception is an agreement on overall scope which includes high level requirements and business cases.

Phase 2 : **Elaboration:** Elaboration is an agreement on design approach that includes baseline architecture describing most of the requirements in detail.

Phase 3 : **Construction:** Construction phase applies an approach in the form of code. Construction designs an actual working product on which testing is also done.

Phase 4 : **Transition:** This phase validates the solution and takes stakeholders acceptance.

8.2.2 Advantages of Iterative Development [S-19]

- Various advantages of iterative development are listed below:
  1. Better productivity, less project failure and lower defect rates; shown by research into iterative and evolutionary methods.
  2. User engagement early feedback and adaptations, leading to a refined system that more closely meets the real needs of the stakeholders.
  3. The learning within iteration can be methodically used to improve the development process itself, iteration by iteration.
  4. Early visible progress.
  5. Managed complexity, the team is not overwhelmed by "analysis paralysis" or very long and complex steps.
  6. Early rather than late mitigation of high risks requirements, technical, usability, objectives etc.

8.3 UNDERSTANDING REQUIREMENTS [W-18]

- Requirements are capabilities and conditions to which the system and the project must conform.
- Primary challenge of requirements work is to find, communicate, and record what is needed, in a form that clearly speaks to the client and development team members.
- The UP promotes a set of best practices, one of which is management of requirements.
- The requirement can be defined as, "a condition needed by a user to solve a problem or achieve an objective/goal".
- Requirements management is a systematic approach to eliciting, organizing, communicating and managing the changing requirements of a software-intensive system or application.
- The benefits of effective requirements management include the following:
  1. Improved software quality and customer satisfaction.
  2. Better control of complex projects.
  3. Improved team communication.
  4. Reduced project costs and delays.

Types and Categories of Requirements:

- In the UP, requirements are categorized according to the FURPS + model [Grady 92].
  1. **Functional:** IEEE defines functional requirements as "a function that a system must be able to perform". It includes features, capabilities, and security.
  2. **Usability:** Describe the case in which users are able to operate the software like human factors, help, and documentation.
  3. **Reliability:** Describe the acceptable failure rate including frequency of failure, recoverability, and predictability.
  4. **Performance:** Describe requirements like response times, throughput, accuracy, availability, resource usage.
  5. **Supportability:** Describe requirements like adaptability, maintainability, internationalization, configurability.
- The "+" in FURPS indicates ancillary and sub-factors, such as:
  1. **Implementation:** Describe requirements like resource limitations, language and tools, hardware etc.
  2. **Interface:** Constraints imposed by interfacing with external systems.
  3. **Operations:** System management in its operational setting.
  4. **Packaging:** For example, a physical box.
  5. **Legal:** Licensing and so forth.
- Some of these requirements are collectively called the 'quality attributes', 'quality requirements' of a system. These include usability, reliability, performance and supportability. In common usage, requirements are categorized as **functional** or **non-functional**. Some dislike this broad generalization, but it is very widely used.

Other Requirements:

1. The **Vision** summarizes the "vision" of the project, an executive summary. It serves to quickly communicate the big ideas.
2. The **Glossary** captures terms and definitions; it can also play the role of a data dictionary.
3. The **Business Rules** (or Domain Rules) capture long-living and spanning rules or policies, such as tax laws, that transcend one particular application.
4. The **Supplementary Specification** captures and identifies other kinds of requirements, such as reports, documentation, packaging, supportability, licensing and so forth.

8.4 UNIFIED PROCESS (UP)

- Unified Process (UP) is a popular iterative and incremental software development process framework.
- The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process (RUP). Rational Unified Process (RUP) is a software development process for object-oriented models.

- It is proposed by Ivar Jacobson, Grady Booch, and James Rambaugh.
- The Unified Process is a traditional "cathedral" style of incremental design driven by constructing views of system architecture. It has the following key features:
  1. UP is component based, commonly being used to coordinate object oriented programming projects.
  2. UP uses UML - a diagrammatic notation for object oriented design for all blueprints.
  3. The design process is anchored and driven by use-cases which help keep sight of the anticipated behaviours of the system.
  4. UP is architecture centric.
  5. Design is iterative and incremental via a prescribed sequence of design phases within a cycle process.
- The Unified Process fits the general definition of a process: " a set of activities that a team performs to transform a set of customer requirements into a software system".
- UP is also referred to as the unified software development process. It describes an approach to building, deploying and possibly maintaining software.
- The Unified Process (UP) is a relatively popular iterative process for projects using OOA. So, it is common and promotes widely recognized best practices. It is also useful for industry professionals to know it and students entering the workforce to be aware of it.

8.4.1 UP Phases

- The Unified Process is an iterative and incremental development process. The Elaboration, Construction and Transition phases are divided into a series of time boxed iterations.
- Each iteration results in an increment, which is a release of the system that contains added or improved functionality compared with the previous release.

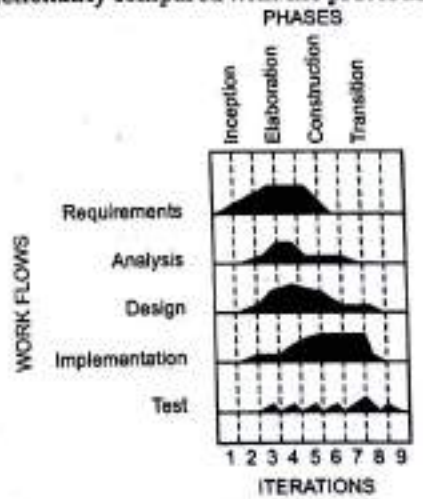


Fig. 8.5: Unified process Phases and Workflows

- It has four sequential phases: Inception, Elaboration, Construction, and Transition. Each of them plays a vital role in managing iterative and incremental development projects using RUP. Each phase concludes with a major milestone.

1. Inception: [S-18, W-18, S-19]

- Inception means start. Inception is the point where the project is proposed. The outcome of inception is project plans, approximate vision, business case, scope, indefinite estimates.
- Inception is the smallest phase in the project, and ideally it should be quite short. If the inception phase is long then it may be an indication of excessive up-front specification, which is contrary to the spirit of the Unified Process.
- The following are primary goals for the inception phase.
  - Establish a justification or business case for the project.
  - Establish the project's scope and boundary conditions.
  - Outline the use cases and key requirements that will drive the design tradeoffs.
  - Outline one or more candidate architectures.
  - Identify risks.
  - Prepare a preliminary project schedule and cost estimate.

2. Elaboration:

- Elaboration means refinement. The outcome of the elaboration is refined vision, iterative implementation of the core architecture, resolution of high risks, identification of most requirements and scope, more realistic estimates.
- During the Elaboration phase the project team is expected to capture a strong majority of the system requirements. However, the primary goals of Elaboration are to report known risk factors and to establish and validate the system architecture.
- The following are primary goals for the inception phase:
  - To establish design and architecture (e.g., using basic class diagrams, package diagrams, and deployment diagrams).
  - To capture a majority of the system's requirements (e.g., in the form of use cases).
  - To perform identified risk analysis and make a plan of risk management to reduce or eliminate their impact on the final schedule and product.
  - To create a plan (schedule, cost estimates, and achievable milestones) for the next (construction) phase.

3. Construction:

- Construction is a manufacturing process. Construction means to build. This phase constitutes detailed design and construction of source code. Iterative implementation of the remaining lower risk and easier elements and preparation for deployment.
- In this stage, the development of the project is completed. The application design is finished and the source code is written. It is in this stage that the software is tested to determine if the project has met its goal laid out in the inception phase.

- The primary objective is to build the software system. In this phase, the main focus is on the development of components and other features of the system. This is the phase when the bulk of the coding takes place. In larger projects, several construction iterations may be developed in an effort to divide the use cases into manageable segments that produce demonstrable prototypes.

4. Transition:

- Transition means delivery. Transition phase consists of development i.e. delivery of the system to the user community. The outcome of this phase is beta testing deployment.
- The final project phase is Transition. In this phase the system is deployed to the target users. Feedback received from an initial release (or initial releases) may result in further refinements to be incorporated over the course of several Transition phase iterations. The Transition phase also includes system conversions and user training.
- In this stage, any fine-tuning is performed. Any final adjustments can be based on user feedback, usability or installation issues.
- The primary objective is to 'transit' the system from development into production, making it available and understood by the end user. The activities of this phase include training the end users and maintainers and beta testing the system to validate it against the end users' expectations. The product is also checked against the quality level set in the Inception phase.
- If all objectives are met, the product release milestone is reached and the development cycle is finished.

8.4.2 Five Phase Workflows of Use Case Model

- Within the Unified Process, five workflows cut across the set of five phases i.e., Requirements, Analysis, Design, Implementation and Test. Each phase in RUP has a workflow, which describes the sequence in which activities from across various disciplines can be performed to achieve the objectives of the respective phase milestone.
- The following sections provide brief overviews of these workflows as shown in Fig. 8.6.

1. Requirements:

- The primary activities of the Requirements workflow are aimed at building the use case model, which captures the functional requirements of the system being defined.
- This model helps the project stakeholders reach agreement on the capabilities of the system and the conditions to which it must conform.
- The use case model also serves as the foundation for all other development work.
- Fig. 8.6 shows how the use case model influences the other five models.

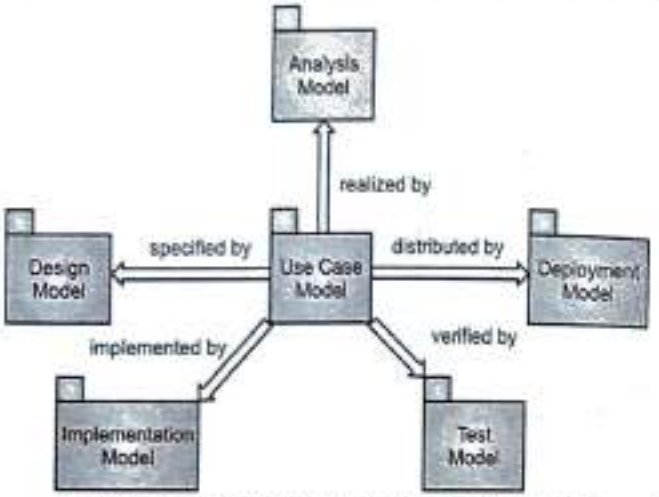


Fig. 8.6: Use case Model influences the other five Models

2. **Analysis:**

- The primary activities of the Analysis workflow are aimed at building the analysis model, which helps the developers refine and structure the functional requirements captured within the use case model.
- This model contains realizations of use cases that lend themselves to design and implementation work better than the use cases.

3. **Design:**

- The primary activities of the Design workflow are aimed at building the design model, which describes the physical realizations of use cases from the use case model and also the contents of the analysis model. The design model serves as an abstraction of the implementation model.
- The design workflow also focuses on the deployment model, which defines the physical organization of the system in terms of computational nodes.

4. **Implementation:**

- The primary activities of the implementation workflow are aimed at building the implementation model, which describes how the elements of the design model are packaged into software components, such as source code files, dynamic link libraries (DLLs) and EJBs.

5. **Test:**

- The primary activities of the Test workflow are aimed at building the test model, which describes how integration and system tests will exercise executable components from the implementations. The test model also describes how the team will perform those tests as well as units tests.

- The test model contains test cases that are often derived directly from use cases. Testers perform black-box testing using the original use case text and white-box testing of the realizations of those use cases, as specified within the analysis model. The test model also contains the results of all levels of testing.

8.4.3 Use Case Model from Inception to Elaboration

- The functionality provided by the system is documented in a use case model that illustrates the system's intended functions (use cases), its surroundings (actors), and the relationships between the use cases and actions (use case diagrams).
- The most important role of a use case model is to provide a vehicle used by the customers or end users and the developers to discuss the system's functionality and behaviour.
- The use case model starts in the Inception Phase with the identification of actors and principal use cases for the system. The model is then matured in the Elaboration Phases.

Summary

- Object-oriented analysis is a process that groups items that interact with one another, typically by class, data or behavior, to create a model that accurately represents the intended purpose of the system as a whole.
- The primary tasks in Object-Oriented Analysis (OOA) are:
  Find the objects, Organize the objects, Describe how the objects interact, Define the behavior of the objects, Define the internals of the objects.
- Iterative process is a process for arriving at a decision or a desired result by repeating rounds of analysis or a cycle of operations.
- In iterative development, a process is broken into a series of steps or iterations.
- Evolutionary development is an iterative and incremental approach to software development.
- Iterative development contains four major phases: Inception, Elaboration, Construction, and Transition.
- The Unified Process is a traditional "cathedral" style of incremental design driven by constructing views of system architecture.
- Unified Process is architecture centric.
- Within the Unified Process, five workflows cut across the set of five phases i.e., Requirements, Analysis, Design, Implementation and Test. Each workflow is a set of activities that various projects workers perform.

Check Your Understanding

1. Which answer below is not one of the RUP life cycle phases?
   (a) Inception Phase (b) Iteration Phase
   (c) Elaboration Phase (d) Transition Phase

2. The RUP is normally described from three perspectives-dynamic, static & practice. What does static perspective do?
   (a) It shows the process activities that are enacted
   (b) It suggests good practices to be used during the process
   (c) It shows the phases of the model over time
   (d) All of the mentioned
3. RUP stands for _____ created by a division of _____.
   (a) Rational Unified Program; IBM
   (b) Rational Unified Process; Infosys
   (c) Rational Unified Process; Microsoft
   (d) Rational Unified Process; IBM
4. Which phase of the RUP is used to establish a business case for the system?
   (a) Transition (b) Elaboration
   (c) Construction (d) Inception
5. Which one of the following is a functional requirement?
   (a) Maintainability (b) Portability
   (c) Robustness (d) None of the mentioned
6. Which one of the following is a requirement that fits in a developer's module?
   (a) Availability (b) Testability
   (c) Usability (d) Flexibility
7. Iterative, incremental model is proposed by _____.
   (a) OD (b) SQL
   (c) UP (d) UML
8. According to components of FURPS+, which of the following does not belong to S?
   (a) Testability (b) Speed Efficiency
   (c) Installability (d) Serviceability
9. Why is Requirements Elicitation a difficult task?
   (a) Problem of scope (b) Problem of understanding
   (c) Problem of volatility (d) All of the mentioned
10. Phase that delivers the software increment and assesses work products that are produced as end users work with software is _____.
    (a) transition (b) elaboration
    (c) construction (d) inception

Answers

1. (a)	2. (a)	3. (d)	4. (d)	5. (d)	6. (b)	7. (c)	8. (b)	9. (d)	10. (a)

Practice Questions

Q.I Answer the following questions in short.

1. What is meant by Object Oriented Analysis?
2. Define an elaboration phase during analysis.
3. Which are common models used in OOA?
4. What is a requirement?
5. What is the workflow detail?
6. What are the "5 Disciplines" in the iterative life cycle?
7. In which phase of the "RUP Process" is the software accepted by the end users?
8. In which phase of Rup is the "Business Case" for the project established?

Q.II Answer the following questions.

1. Write a note on Rational Unified Process.
2. Describe five workflows of Unified Process
3. What is evolutionary and iterative development?
4. What is Iterative Development? List phases of Iterative Development.
5. With the help of a diagram describe UP phases.
6. What is the requirement? Explain with its types.

Q.III Define the terms.

1. Unified Process
2. Inception
3. Transition
4. Workflows
5. Iteration

Previous Exam Questions

Summer 2018

1. Define Inception. [2 M]

Ans. Refer to Section 8.4.2

2. What is UP? Explain any two phases in detail. [4 M]

Ans. Refer to Sections 8.4.1 and 8.4.2.

Winter 2018

1. What is Inception? [2 M]

Ans. Refer to Section 8.4.2.

2. Describe UP phases with the help of diagrams. [4 M]

Ans. Refer to Sections 8.4.1 and 8.4.2.

3. Explain Understanding Requirement of Object Oriented Analysis. [4 M]

Ans. Refer to Section 8.3.

Summer 2019

1. What is meant by inception? **2 M**

Ans. Refer to Section 8.4.2.

2. Define UP. Explain any two phases in detail. **4 M**

Ans. Refer to Sections 8.4.1 and 8.4.2.

3. Explain five UP workflows in UP in detail. **4 M**

Ans. Refer to Section 8.4.2.

4. What is meant by Iterative Development? State its various advantages. **4 M**

Ans. Refer to Sections 8.2 and 8.2.2.

■■■

9... Object Oriented Design

Learning Objectives ...

- To understand the Object-oriented designing concept in detail.
- To understand various Object-oriented methods in detail.
- To understand system designing methods.

9.1 INTRODUCTION

- Object Oriented Design (OOD) involves implementation of the conceptual model produced during object-oriented analysis.
- In OOD, concepts in the analysis model, which are technology independent, are mapped onto implementing classes, constraints are identified and interfaces are designed, resulting in a model for the solution domain i.e., a detailed description of how the system is to be built on concrete technologies.
- The implementation details generally include:
  1. Restructuring the class data (if necessary).
  2. Implementation of methods, i.e., internal data structures and algorithms.
  3. Implementation of control.
  4. Implementation of associations.
- Grady Booch has defined Object-Oriented Design as "a method of design encompassing the process of object-oriented decomposition and a notation for depicting logical and physical as well as static and dynamic models of the system under design".
- **The characteristics of an Object-Oriented Design (OOD) are:**
  1. In an object-oriented design a software system is designed as a set of interacting objects that manage their own private state and offer services to other objects.
  2. Objects are created by instantiating an object class that defines the attributes and operations associated with the object. In Fig. 9.1, the object name precedes the colon in the top part of the rectangle and the class name follows the colon. Several objects of the same class may co-exist in the same program.

9... Object Oriented Design

Learning Objectives ...

- To understand the Object-oriented designing concept in detail.
- To understand various Object-oriented methods in detail.
- To understand system designing methods.

9.1 INTRODUCTION

- Object Oriented Design (OOD) involves implementation of the conceptual model produced during object-oriented analysis.
- In OOD, concepts in the analysis model, which are technology independent, are mapped onto implementing classes, constraints are identified and interfaces are designed, resulting in a model for the solution domain i.e., a detailed description of how the system is to be built on concrete technologies.
- The implementation details generally include:
  1. Restructuring the class data (if necessary).
  2. Implementation of methods, i.e., internal data structures and algorithms.
  3. Implementation of control.
  4. Implementation of associations.
- Grady Booch has defined Object-Oriented Design as "a method of design encompassing the process of object-oriented decomposition and a notation for depicting logical and physical as well as static and dynamic models of the system under design".
- **The characteristics of an Object-Oriented Design (OOD) are:**
  1. In an object-oriented design a software system is designed as a set of interacting objects that manage their own private state and offer services to other objects.
  2. Objects are created by instantiating an object class that defines the attributes and operations associated with the object. In Fig. 9.1, the object name precedes the colon in the top part of the rectangle and the class name follows the colon. Several objects of the same class may co-exist in the same program.

3. Object classes are abstractions of real-world or system entities that encapsulate state information and that define a number of operations (alternatively called services or methods) that create, access or modify the state.
4. Objects are independent entities that may readily be changed because state and representation information is held within the object. Changes to the representation may be made without reference to other system objects.
5. System functionality is expressed in terms of operations or services associated with each object. Objects interact by calling on the operations defined by other objects. Interaction in Fig. 9.1 is shown by arrows linking the objects.
6. There are no shared data areas. Objects communicate by calling on services offered by other objects rather than sharing variables. There is no possibility that a program component can be affected by modifications to shared information. Changes are therefore easier to implement.
7. Objects may be distributed and may execute either sequentially or in parallel.

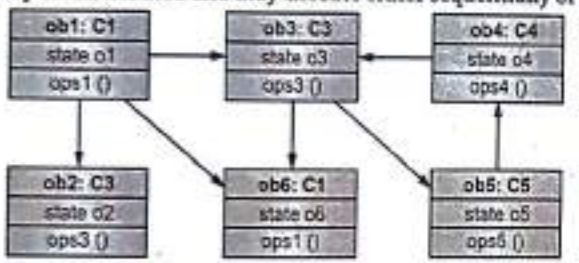


Fig. 9.1: System designed as a number of interacting objects

Design for Object Oriented Systems:

- The design pyramid (shown in Fig. 9.2) focuses exclusively on the design of the specific product or system.

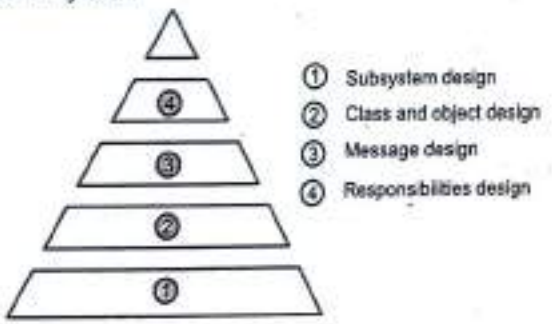


Fig. 9.2: Object Oriented Design Pyramid

- It should be noted, however, that another layer of design exists, and this layer forms the foundation on which the pyramid rests.

- This foundation layer focuses on the design of "domain objects" (called design patterns).
- Domain objects play a key role in building the infrastructure for the object oriented system by providing support for human-computer interface activities, task management, and data management.
- Domain objects can also be used to flush out the design of the application itself.
- Most object oriented design (the four layer pyramid of object oriented design) processes recognize four types of design work. These types are as follows:

1. **The Subsystem design:** It contains a representation of each of the subsystems that enable the software to achieve its customer defined requirements and to implement the technical infrastructure that supports customer requirements.
2. **The Class and Object design:** It contains the class hierarchies that enable the system to be created using generalizations and increasingly more targeted specializations. This layer also contains design representations of each object.
3. **The Message design:** It contains the details that enable each object to communicate with other objects. This layer establishes the external and internal interfaces for the system.
4. **The Responsibilities design:** It contains the data structure and algorithmic design for all attributes and operations for each object.

Design Issues:

- Following are the five criteria for judging the ability of a design method to achieve high quality of object oriented design:

1. **Decomposability:** The facility with which a design method helps the designer to decompose a large problem into subproblems those are easier to solve.
2. **Composability:** The degree to which a design method ensures that program components, once designed and built, can be reused to create other systems.
3. **Understandability:** The facility with which a program component can be understood without reference to other information or other modules.
4. **Continuity:** The ability to make small changes in a program and have these changes displays themselves with corresponding changes in one or a few modules.
5. **Protection:** An architectural characteristic that will reduce the propagation of side effects if an error does occur in a given module.

9.2 OBJECT ORIENTED METHODS

- When the object oriented comes into the picture, most of the methodologists can work on the analysis and design methods and processes would bit into an object-oriented world.
- The period between 1980 and 1995 is very important for the development of the methodologies of the analysis and design of the object-oriented concepts.

- A methodology is a systematic way of doing things.
- Object-oriented methodology is a set of methods, models and rules for developing systems.
- Object Oriented Methodology (OOM) is a methodology for object oriented development and a graphical notation for representing objects oriented concepts.
- Many methodologies have been proposed for object-oriented software development. (See Table 9.1).
- A methodology usually includes:
  1. **A notation:** It is graphical representation of classes and their relationships and interactions.
  2. **A process:** Suggested set of steps to carry out for transforming requirements into a working system.

Table 9.1: The popular methodologies of the object-oriented approach

Methodology	Authors	Year
1. OMT (Object Modeling Technique)	J. Rumbaugh, M. Blaha, W. Premerlany, F. Eddy and W. Lorensen	1991
2. OOD (Object Oriented Design)	Grady Booch	1991
3. OOSE (Object Oriented Software Engineering)	Jacobson, Christerson, Jonson and G. Overgaard	1994
4. OOA (Object Oriented Analysis)	Shlaer and Steve Mellor	1987
5. OOA/OOD	T. Coad and E. Yourdon	1991
6. OOM (Object Oriented Methodology)	M. Bouzeghoub and A. Rochfeld	1993
7. Jackson System/Structured Development (JSD)	Jackson and John Cameron	1983

9.2.1 Booch Method

- The Booch Method is an object-oriented software development method used to analyze, model and document system requirements.
- Grady Booch's Object-Oriented Design (OOD), also known as Object-Oriented Analysis and Design (OOAD), is a precursor to the Unified Modeling (UML).
- The Booch method is a well-known OO method, which helps us in designing a system using objects. It covers the analysis and design aspects of an OO system.

Grady Booch

- G. Booch tries to cover the aimed physical design and programming and ignores the requirement analysis and conceptual design.
- G. Booch uses many diagrams in his Object-Oriented Design at two different levels; one is design level and another is implementation level.
- According to design and implementation levels different diagrams are used by G. Booch in his level which are given below:

Table 9.2: Diagrams used by G. Booch

Sr. No.	Diagram type	Defined Level
1.	Object diagram	Design Level
2.	Class diagram	
3.	State transition diagram	
4.	Interaction diagram	
5.	Process diagram	Implementation level
6.	Module diagram	

- Booch explains what we can do in terms of system definition, but he does not provide us any guidelines regarding what we should do so as to come up with a better analysis and design model of the system.
- The Booch method is classified as a second-generation, object-oriented analysis and design method.
- The G. Booch method encompasses both a "micro-development process", and a "macro-development process".

1. **The Macro Process:**

- The macro process is a high-level process describing the activities of the development team as a whole. It approaches the development process from a manager's perspective.
- Macro process focuses on the two manageable elements one is Risk and the other is Architectural vision. The main objectives of macro processes is the technical management of the system that gives less interest to the actual object oriented design than in how well the project corresponds to the requirements set for it and whether it is produced on time.

- The macro process consists of the steps or phases, shown in the Fig. 9.3 (a).

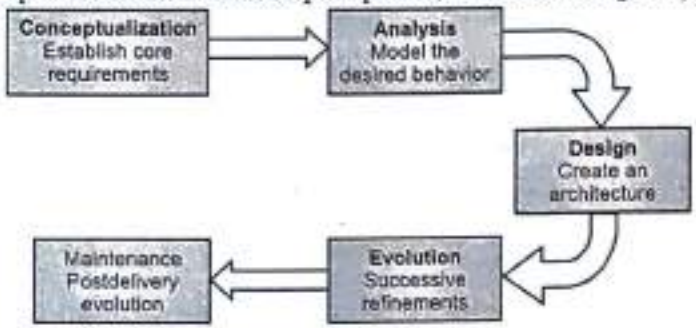


Fig. 9.3 (a): Macro Process

2. The Micro Process:

- The micro process is a lower-level process that represents the technical activities of the development team. It is composed of the steps shown in the Fig. 9.3 (b).
- The micro level defines a set of design tasks that are re-applied for each step in the macro process. Every macro development process has its own micro development process. It defines the daily activities of the single developer. While the macro process serves on a controlling framework for the micro process.
- This process is applied to each architectural release of the system.

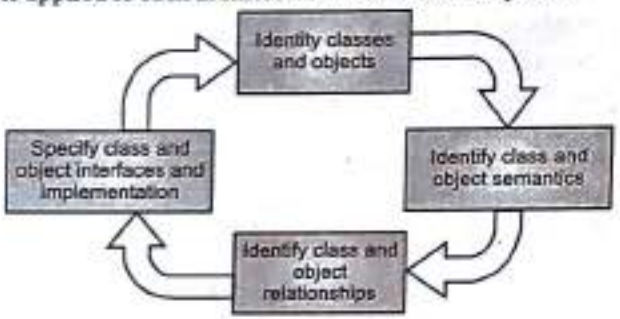


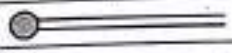
Fig. 9.3 (b): Micro Process

Notations in G. Booch Method:

- Notation plays an important part in any methodology. It holds the process together.
- The Booch method provides a very robust notation, which grows from analysis into design.
- Certain elements of the notation (i.e., classes, association, aggregations, inheritance) are introduced during analysis. Other elements of the notation (i.e. class categories containment indicators and adornments) are introduced during design and evolution.

- Notations have three roles:
  1. Notations provide semantics that are rich enough to capture all important strategic and tactical decisions.
  2. Notations serve as the language for communicating decisions that are not obvious or cannot be inferred from the code itself.
  3. Notations offer a form concrete enough for humans to reason about and for tools to manipulate.
- Following are the notations which are used by G. Booch for the class and various relationships.

Table 9.3: Notations used by G. Booch for Class and Relationship

Shape Used	Purpose
	Class
	Association
	Inheritance, mostly used for specialization
	Has
	Using

- Example of Class Diagram:

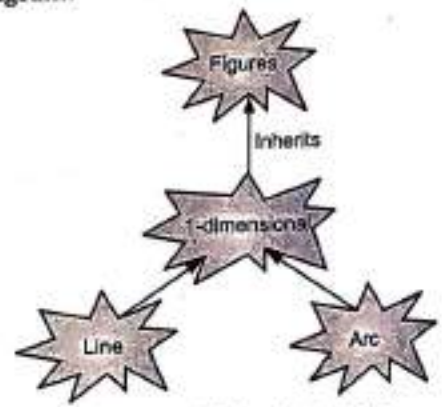


Fig. 9.4: Modeling using Booch

- In the above Class diagram, the arrow represents specialization. For example, the class `Line` as subclass of class `1-dimensional`.

Models to describe an OO system:

- G. Booch defines different models to describe an OO system. These models are:
  1. **Physical model** describes the various processes and modules that the system will have at a later state.
  2. **Logical model** describes the classes and objects in the system. In other words, it describes the problem domain. It discusses the classes and objects in a system.
  3. **Static model** describes how the classes and objects in a system are related to each other.
  4. **Dynamic model** shows how the classes, objects and other aspects of a system work together to produce the desired results.

9.2.2 The Coad and Yourdon Method

- Yourdon and Coad's Object-Oriented Analysis and Design (OOA/OOD) is an Object-oriented Method that precedes UML.

Edward Yourdon

Peter Coad

- This method is only one model that describes the static characteristics.
- Generally, this method is based upon:
  - Information modeling.
  - Object oriented programming language.
  - Knowledge based system.
- Coad – Yourdon method is simply based on the technique known as "SOSAS". (Subjects, Objects, Structures, Attributes Services).
- The Coad and Yourdon method for OOD was developed by studying how "effective object-oriented designers" do their design work.

Major system components:

- The design approach addresses not only the application but also the infrastructure for the application and focuses on the representation of four major system components i.e. the problem domain component, the human interaction component, the task management component, and the data management component which is explained in detail below:

1. **Problem Domain Component:**
   - Group all domain specific classes.
   - Design an appropriate class hierarchy for the application classes.
   - Work to simplify inheritance, when appropriate.
   - Refine design to improve performance.
   - Develop an interface with the data management component.
   - Refine and add low-level objects as required.
   - Review design and challenge any additions to the analysis model.
2. **Human Interaction Component:**
   - Define the human actors.
   - Develop task scenarios.
   - Design a hierarchy of user commands.
   - Refine the user interaction sequence.
   - Design relevant classes and class hierarchy.
   - Integrate GUI classes as appropriate.
3. **Task Management Component:**
   - Design the data structures and layout.
   - Design services required to manage the data structures.
   - Identify tools that can assist in implementing data management.
   - Design appropriate classes and class hierarchy.
4. **Data Management Component:**
   - Design the data structures and layout.
   - Design services required to manage the data structures.
   - Identify tools that can assist in implementing data management.
   - Design appropriate classes and class hierarchy.

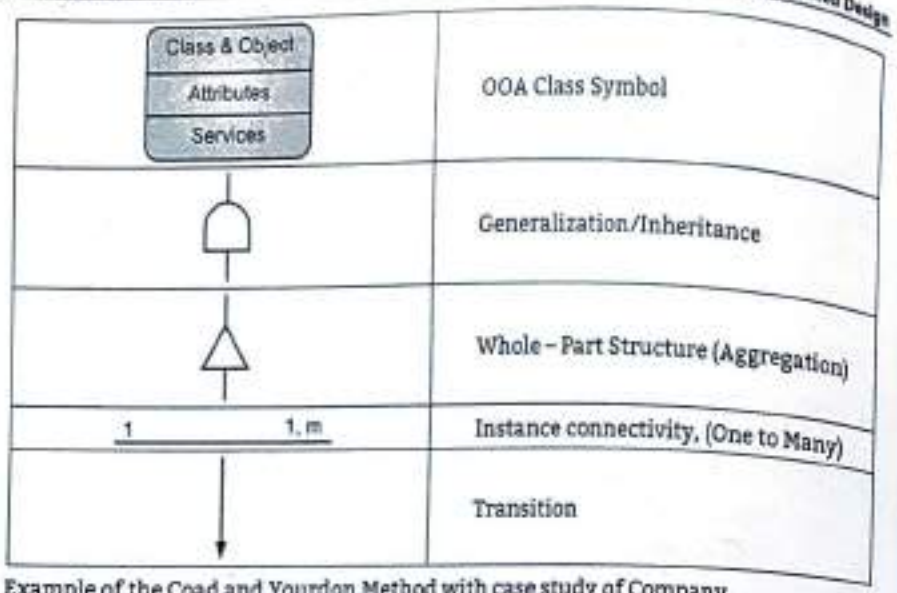
Notations:

- Following are the notations which are used by the Coad – Yourdon for the class and various relationship:

Table 9.4: Notations used by Coad – Yourdon for the Class and Relationship

Shape Used	Purpose
Class & Object Attributes Services	OOA Class and its object

contd. ...



- Example of the Coad and Yourdon Method with case study of Company.

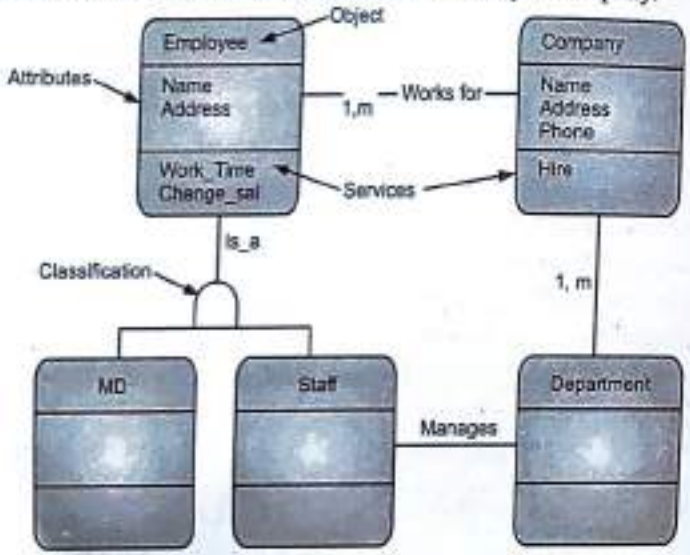


Fig. 9.5: Object modeling using Coad and Yourdon method

9.2.3 Jacobson Method

- The object oriented model of Jacobson describes the development of a sequence of models from requirement to implementation.
- OOSE was developed by Ivar Jacobson in 1992. OOSE is the first object oriented design methodology that employs use cases in software design.

Ivar Jacobson

- The central objective of the Jacobson method is the Use Case concept.
- OOSE (Object Oriented Software Engineering) is a software design technique that is used in software design in object oriented programming.
- The design activity for OOSE is a simplified version of the proprietary objectory method, also developed by Jacobson.
- The design model highlights traceability to the OOSE analysis model. First, the idealized analysis model is adapted to fit the real world environment. Then primary design objects, called blocks, are created and categorized as interface blocks, entity blocks, and control blocks.
- Communication between blocks during execution is defined and the blocks are organized into subsystems.
- OOSE includes a requirement, an analysis, a design, an implementation, and a testing model. The relationship between the central use case diagram and the various models as shown in Fig. 9.6.

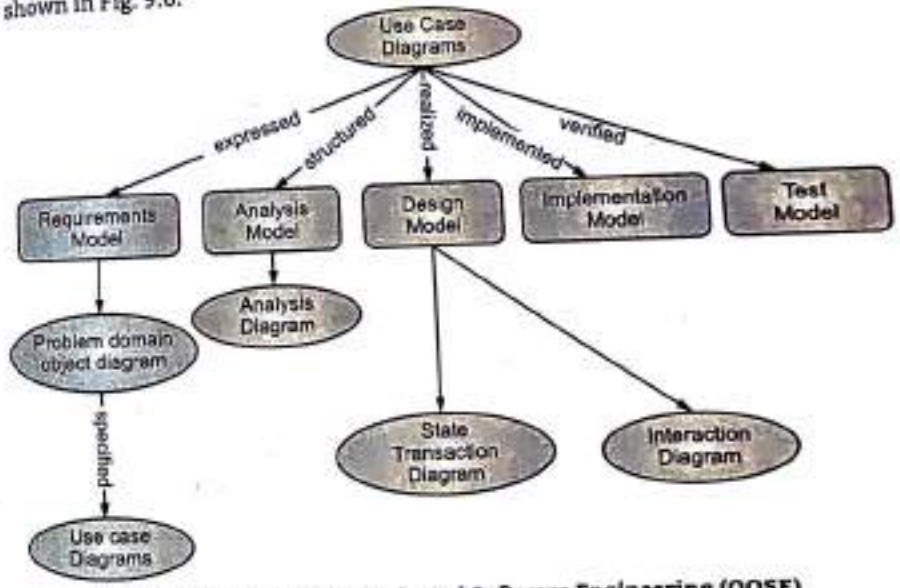


Fig. 9.6: Jacobson's Object-Oriented Software Engineering (OOSE)

- A brief outline of Jacobson's method is as follows:
  - Consider adaptations to make the idealized analysis model fit the real world environment.
  - Create blocks as the primary design object.
  - Create an interaction diagram that shows how stimuli are passed between blocks.
  - Organize blocks into subsystems.
  - Review the design work.
- Although the terminology and process steps for each of these object oriented design methods differ, the overall object oriented design processes are reasonably consistent.
- To perform object oriented design, a software engineer should perform the following generic steps:
  1. Describe each of the subsystems in a manner that is implementable.
  2. Object design
  3. Message design
  4. Review the design model and iterate if needed.

Notations:

- Following are the notations which are used by the Jacobson method:

Table 9.5: Notations used by the Jacobson Method

Shape Used	Purpose
	Actor
	Use Case
	Communication
	Boundary

- Fig. 9.7 shows an example of the Jacobson method.

Fig. 9.7: Example of the Jacobson Method.

9.2.4 Rumbaugh Method [S-19]

- James Rambaugh found out the method of the Object Modeling Technique (OMT).
- OMT describes the analysis, design and implementation of a system.
- He used familiar symbols and techniques in his analysis and design.

James Rumbaugh

- The object modeling technique encompasses a design activity that encourages design to be conducted at two different levels of abstraction.
- System design focuses on the layout for the components that are needed to construct a complete product or system.
- The analysis model is partitioned into subsystems, which are then allocated to processors and tasks.
- A strategy for implementing data management is defined and global resources and the control mechanisms required to access them are identified.
- Object design emphasizes the detailed layout of an individual object. Operations are selected from the analysis model and algorithms are defined for each operation.
- Data structures that are appropriate for attributes and algorithms are represented.
- Classes and class attributes are designed in a manner that optimizes access to data and improves computational efficiency.
- A messaging model is created to implement the object relationships (associations).
- J. Rumbaugh methods consists of following five steps:

Step 1 : **Identifying class and object:** Specifies how classes and objects should be found.

Step 2 : **Identifying structures:** It is done in two different ways:
(i) Generalization: Specification structure, and
(ii) The Whole-Part Structure.

Step 3 : **Identifying subjects:** It is done by partitioning the class and objects models into larger units. Subject is a group of classes and objects.

Step 4 : **Defining attributes:** It is done by identifying information and the associations that should be associated with each and every instance. The identified attributes are placed in the correct level of inheritance hierarchy.

Step 5 : **Defining Services:** It means defining the operations of the classes.

Types of Models:

- OMT proposes three main types of models for creating a blueprint of a software project. These models are: Object model, Dynamic model, and Functional model.
- J. Rumbaugh provides three different views of system using three models as given below:

1. **Object Model:**
- Object model describes the static structure of the objects in the system, identification, attributes, operations and their relationship. It also deals with structural data portions of a system.

- Fig. 9.8 shows steps in object modeling.

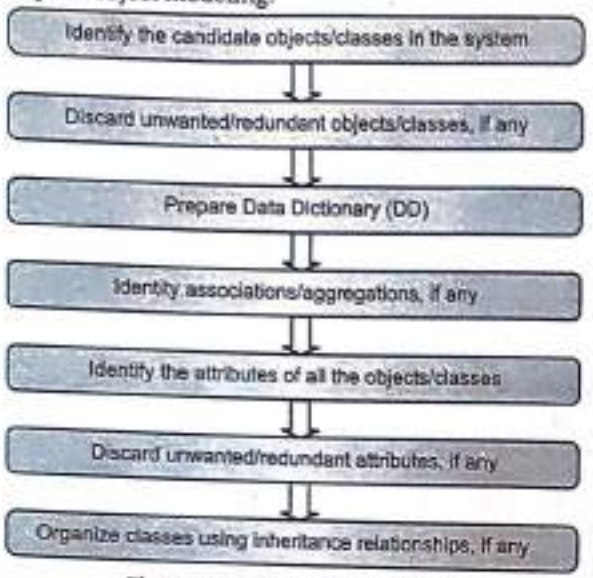


Fig. 9.8: Steps in Object Modeling

- The main concepts of object model are:
  - Class
  - Attributes
  - Operation
  - Inheritance
  - Association (i.e. relationship)
  - Aggregation
- The Rumbaugh object model is very much like an entity relationship diagram except that there are now behaviours in the diagram and class hierarchies.

Notations for Object Model:

Table 9.6: Notations for Object Model

Shape Used	Purpose
Class / Attributes / Operation	Class Representation
———	One-to-one association
———●	One-to-many association
———▶	Specialization

Example:

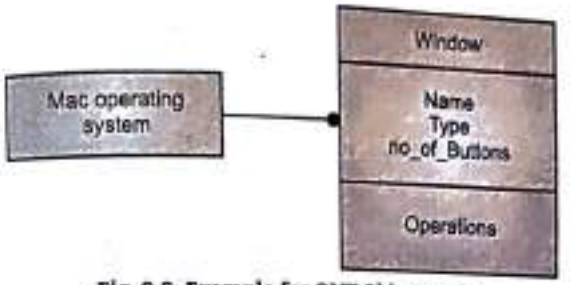


Fig. 9.9: Example for OMT Object Model

2. **Dynamic Model:**
- The dynamic model of Rumbaugh is a "state transition" diagram that shows how an entity changes from one state to another state.
- The dynamic model is related to a time-dependent behaviour and objects of a system.
- The dynamic model is important from the context of interactive systems, but not from static processes, such as database operations. Here, the sequence of interactions is important.
- Main concepts of the dynamic model are states, transitions between states and events to trigger transitions. Actions can be modeled as occurring within states. Generalization and Aggregation, (concurrency) are predefined relationships.
- The broad level steps in a dynamic model are shown in Fig. 9.10.

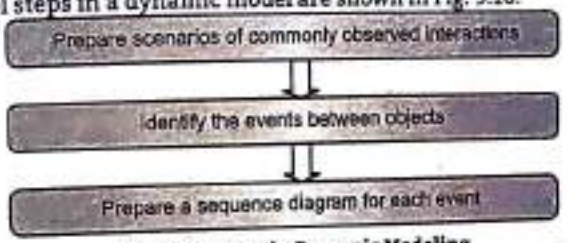


Fig. 9.10: Steps in Dynamic Modeling

- The main aspects of dynamic model are:
  - State
  - Sub/Super state
  - Event
  - Action
  - Activity

Notations:

Table 9.7: Notations for Dynamic Model

Shape Used	Purpose
●——▶	Start
▭	States
——▶	Transition

Example:

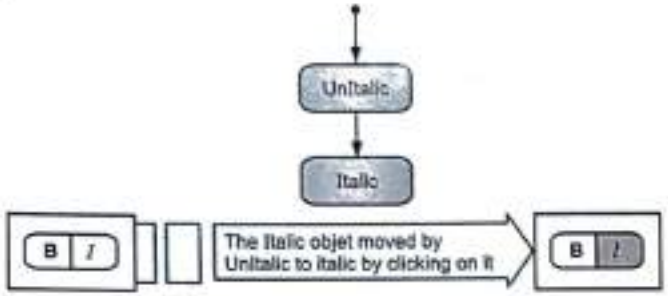


Fig. 9.11: Example of OMT Dynamic Model

3. Functional Model:

- The functional model of Rumbaugh is the equivalent of the familiar Data Flow Diagrams (DFD) from a traditional systems analysis.
- Functional model describes the data value transformation within the system.
- In a functional model, we concentrate on the computations in the system with no regard to the sequence of events, any decisions to be taken, or the structure of the objects.
- Main concepts of functional model are Process, Data store, Data flow and Actors.
- The broad level steps in functional modeling are outlined in Fig. 9.12.

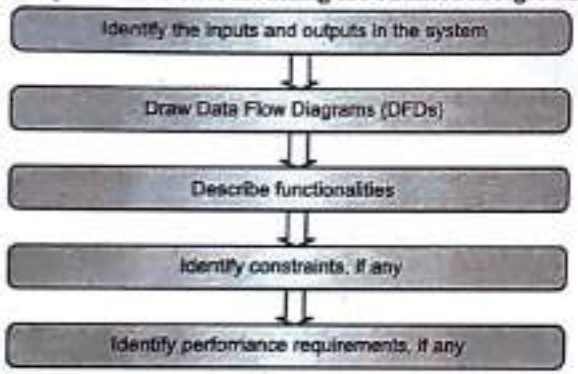


Fig. 9.12: Steps in Functional Modeling

- The main concepts of functional model are:
  - Process
  - Data Store
  - Data Flow
  - Control Flow
  - Actor (Source/Sink)

Notations:

Table 9.8: Notations for Functional Model

Shape Used	Purpose
	Process
	Data Flow
	Data Store
	External Entity

- Fig. 9.13 shows an example of a functional model.

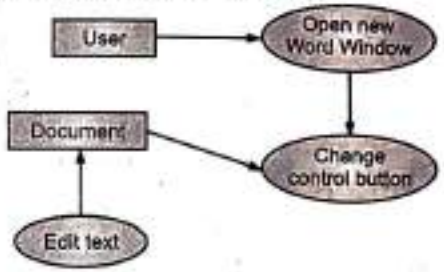


Fig. 9.13: Example of functional model

9.3 THE GENERIC COMPONENTS OF THE OBJECT ORIENTED DESIGN MODEL [S-19]

- The object-oriented model as shown in Fig. 9.14, (Object-Oriented Design Model) builds integration of existing software modules into the system development. A database of reusable components supplies the components for reuse.
- The object-oriented model starts with the formulation and analysis of the problem. The design phase is followed by a survey of the component library to see if any of the components can be re-used in the system development.
- If the component is not available in the library then a new component must be developed, involving formulation, analysis, coding and testing of the module.
- The new component is added to the library and used to construct the new application.
- This model aims to reduce costs by integrating existing modules into development. These modules are usually of a higher quality as they have been tested in the field by other clients and should have been debugged.
- The development time using this model should be lower as there is less code to write.

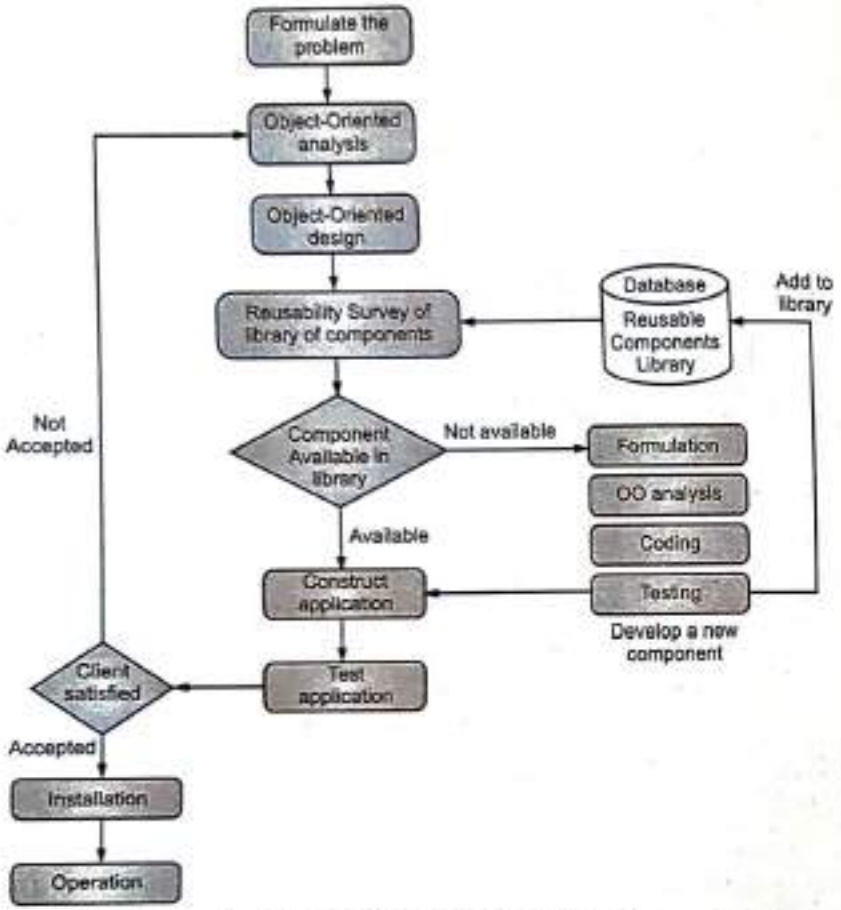


Fig. 9.14: The Object-Oriented Design Model

- The object-oriented model should provide advantages over the other models as the library of components grows over time.
- Once a reasonably complete analysis model has been developed, the software engineer concentrates on the design of the system.
- This is accomplished by describing characteristics of the subsystem required to implement both customer requirements and the support environment that is necessary for their realization.
- As subsystems are defined, they must be coordinated within the overall context of customer requirements.

Design Components:

- During subsystem design, it is necessary for the software engineer to define four important design components:
  1. **Problem domain component:** The subsystems, those are responsible for implementing customer requirements directly.
  2. **Human interaction component:** The subsystems that implement the user interface (this included reusable GUI subsystems).
  3. **Task management component:** The subsystems that are responsible for controlling and coordinating concurrent tasks that may be packaged within a subsystem or among different subsystems.
  4. **Data management component:** The subsystem that is responsible for the storage and retrieval of objects.
- Each of these generic components may be modeled (during object oriented analysis) with a series of classes as well as requisite relationships and behaviours.
- In addition, the design components are implemented by defining the 'protocol' that formally describes the messaging model for each of the components.

9.4 THE SYSTEM DESIGN PROCESS [S-19]

- System design develops the architectural detail required to build a system.
- A system is an organized relationship among functioning units or components to achieve particular goals or objectives".
- System design can be defined as, "the process of taking a logical model of a system together with a strongly stated set of objectives/goals for that system and producing the specifications of a physical system that will meet those objectives/goals."
- Systems design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements.
- The main goal of OOSE is the efficiency, reliability, reusability and the capability of sharing the resources in software systems.
- During the system design process, the developer/programmer/analyst makes planned decisions with broad consequences.
- The architecture of the system is a high level plan or we say strategy for solving application problems.
- Best architectural plan is implemented by user requirements as well as previous experiments.
- System design develops the architectural detail required to build a system or product.

Activities of the system design process:

- The system design process includes the following activities:
  1. Partition the analysis model into subsystems.
  2. Identify concurrency that is dictated by the problem.

3. Allocate subsystems to processors and tasks.
4. Develop a design for the user interface.
5. Choose a basic strategy for implementing data management.
   (i) Identify global resources and the control mechanisms required to access them.
   (ii) Design an appropriate control mechanism for the system, including task management.
   (iii) Consider how boundary conditions should be handled.
   (iv) Review and consider trade-offs.

- In Fig. 9.15, we have seen that process flow starts from Object Oriented Analysis and ends with System Design.

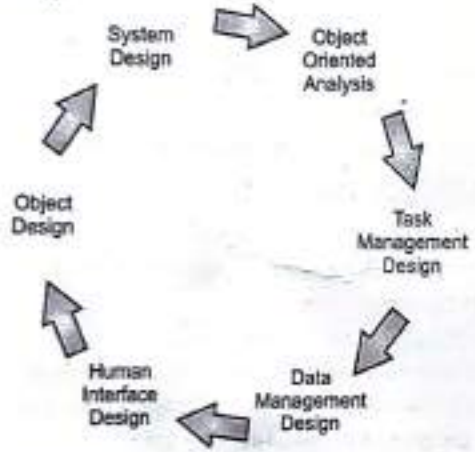


Fig. 9.15: Process flow of Object Oriented Design

- To design the system structure, the previously and analyzed model can be implemented further.

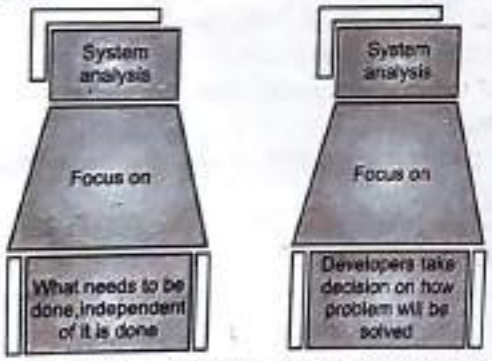


Fig. 9.16: Focus on the System Analysis and Design Phase

- During the system design process, it solves the problems which basically arises from the system analysis Phase, developers thinks on:
  1. Subsystem of the system.
  2. Overall style of the system.
  3. Reuse of the system.
  4. Overall structure of the system.

9.4.1 Partitioning the Analysis Model

- One of the fundamental analysis principles is partitioning. In OO system design, we partition the analysis model to define cohesive collections of classes, relationships and behavior. These design elements are packaged as a subsystem.
- The partitioning in the analysis model of the target system into subsystems, based on the combination of knowledge of the problem domain and the proposed architecture of the target system.

System design process:

1. Breaking a system into sub systems:

- The first step of the system design is breaking up the system into the small pieces of the subsystems. Since, the major piece which further implement is nothing but the subsystem.
- A subsystem is a group of classes, associations, operations, events and constraints that are interrelated and have a well-defined and small interface with other subsystems.
- It should have a well-defined interface through which all communication with the rest of the system occurs. A subsystem is usually identified by the services it provides.

2. The decomposition of systems:

- The decomposition of a system into subsystems (small pieces) may be organized as a sequence of horizontal layers or vertical partitions.
- The number of subsystems should be kept small and it may in turn be decomposed into smaller subsystems of its own. The lowest level subsystems are called **modules**.
- A subsystem of the system can be partitioned internally to help reduce complexity.

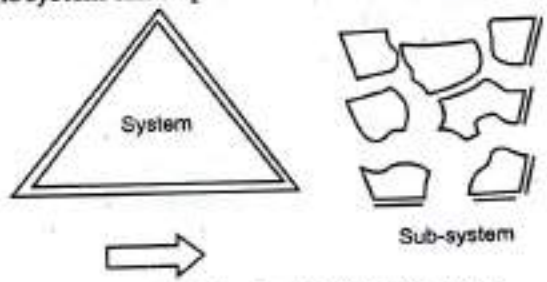


Fig. 9.17: Partitioning the Analysis Model

9.4.2 Concurrency and Sub System Allocation [S-19]

- The dynamic aspect of the object-behavior model provides an indication of concurrency among classes (or subsystems). If classes (or subsystems) must act on events asynchronously and at the same time, they are viewed as concurrent.
- When subsystems are concurrent, two allocation options exist:
  - Allocate each subsystem to an independent processor.

    Or
  - Allocate the subsystems to the same processor and provide concurrency support through operating system features. Concurrent tasks are defined by examining the state diagram for each object.
- Numbers of real world problems as well as in the case of hardware, the objects are concurrent. For example, we work with printers, speakers, keyboard and so on hardware objects work together.
- This is all about hardware objects but this cannot be followed with all software objects. It is not always concurrent because one process supports many objects and hence a single object cannot be utilized at the same time.
- In hardware, two objects are inherently concurrent if they receive events at the same time without interacting. When the events are unsynchronized, the object cannot fold onto a single thread of control. For example, CPU and speakers on a computer must operate concurrently.
- A **thread of control** is a path through a set of state diagrams on which only a single object at a time is active. The thread passes to the receiver of the event until it eventually returns to the original object. The thread splits if the object sends an event and continues executing.
- On each thread of control, only a single object at a time is active, we can implement threads of control as tasks in the computer system.

9.4.3 Task Management Component [S-16]

- P. Coad and E. Yourdon suggest the following strategy for the design of the objects that manage concurrent tasks:
  1. The characteristics of the task are determined.
  2. A coordinator task and associated objects are defined.
  3. The coordinator and other tasks are integrated.
- The characteristics of a task are determined by understanding how the task is initiated. Event-driven and clock-driven tasks are the most commonly encountered.
- Concurrent system must allocate each concurrent subsystem to a hardware unit, either a general purpose processor unit as follows:
  1. Choose software or hardware implementation for subsystems.
  2. Estimating the performance needs the resources needed to satisfy them.

  3. Determine the connectivity of the physical units that implement the subsystem.
  4. Allocate software subsystems to processors to satisfy performance needs and minimize inter-processor communication.
- The basic task template (for a task object) takes the form:
  - **Task name:** The name of the object.
  - **Description:** A narrative describing the purpose of the object.
  - **Priority:** Task priority (e.g. low, medium, high).
  - **Services:** A list of operations that are responsibilities of the object.
  - **Coordinates:** The manner in which object behavior is invoked.
  - **Communicates:** Input and output data values relevant to the task.
- Above template description can then be translated into the standard design model (incorporating representation of attributes and operations for the task objects).

1. **Estimating Hardware Resource Requirement:**

- When a single unit can give less performance and then only we can think about the more units in action. In the case of computers, a single CPU's performance is less than when we go for use of hardware units which are based on higher performance.
- These high performance processor depends on following things:
  (i) Processing time.
  (ii) Output of machine.
  (iii) Volume of computation.
  (iv) Speed of machine.

2. **Determine Physical Connectivity:**

- Choose the topology for connecting the physical units i.e. when we acquire LAN, choose suitable topology like bus, star etc.
- Choose the form of the connection channels and the communication protocols.
- Choose the topology of repeated units.

3. **Allocating Tasks to Processors:**

- There are various different reasons by which system design must allocate tasks for the various software subsystem to processor, they are given below:
  (i) The response time exceeds the available communication bandwidth between a task and piece of hardware.
  (ii) When we used highly interactive subsystems to the same processor then we minimized the communication cost.
  (iii) Certain tasks are required at specific physical locations because of to control hardware or to permit independent operations.
  (iv) To support the task several processors must use since communication rates are too great for a single processor.

4. **Making Hardware – Software Trade Off:**

- In Object Oriented Designing, the duty of the system designer is thinking about which hardware is suitable for the implementation.

9.4.4 The Data Management Component [S-16]

- Data management encompasses two different areas of concern:
  1. The management of data that are critical to the application itself and
  2. The creation of an infrastructure for storage and retrieval of objects.
- In general, data management is designed in a layered fashion. The idea is to isolate the low-level requirements for manipulating data structures from the higher-level requirements for handling system attributes.
- There are various different ways to store the data either separately or in the combination:
  1. Data Structures
  2. Files
  3. Databases
- Like tasks trade off the same factors such as time, cost, capacity, reliability etc. to be considered in case of data management.
  1. **Files are:**
     - Cheap, simple and permanent.
  2. **File operations are:**
     - Applications vary for different processors.
     - Low level.
  3. **Databases managed by DBMS:**
     - Better performance than files.
     - Cost comparatively high with files.
     - Always portable.

9.4.5 The Resource Management Component [W-16]

- A variety of different resources are available to an OO system or product; and in many instances, subsystems compete for these resources at the same time.
- This is the primary duty of the system designer to:
  1. Identify the global resource.
  2. Determine mechanisms for controlling access to them.
- There are several global units such as physical units (such as processors, tape drives, communication satellites etc.), Space (such as disk space, workstation screens etc.), Logical names (such as Object ID, file names, class names etc.) and databases.
- When the resource is a physical object, then it can control itself by establishing a protocol for obtaining access.
- When the resource is a logical entity, such as an object ID or a database, then there is danger of conflicting access in a shared environment.

9.4.6 Inter Sub System Communication

- Once each subsystem has been specified, it is necessary to define the collaborations that exist between the subsystems. The model that we use for object-to-object collaboration can be extended to subsystems as a whole.
- Communication can occur by establishing a Client/Server link or a Peer-to-Peer link. Referring to the Fig. 9.18, we must specify the contract that exists between subsystems. Recall that a contract provides an indication of the ways in which one subsystem can interact with another.

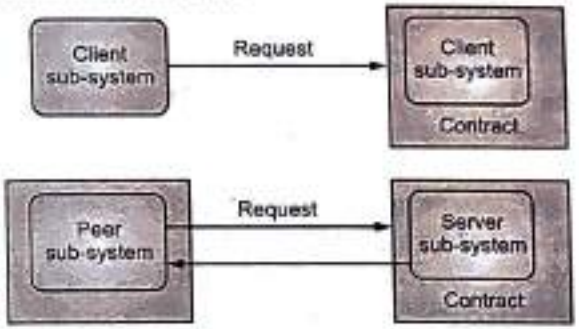


Fig. 9.18: A model of collaboration between subsystems

- Interactive interface i.e., the communication protocol between the one sub system with another subsystem.

Communication Issues:

- The major issues of the inter subsystem communication are as follows:
  - User interface
  - Flow control within system
  - Proper Syntax
  - Performance and error handling
  - Output format

9.5 OBJECT DESIGN PROCESS

- Object design is concerned with the detailed design of the objects and their interactions.
- It is completed within the overall architecture defined during system design and according to agreed design guidelines and protocols.
- Object design is particularly concerned with the specification of attribute types, how operations function, and how objects are linked to other objects.

1. **Object Descriptions:**

- A design description of an object (an instance of a class or subclass) can take one of two forms as given below:

(i) **A Protocol description:** This description establishes the interface of an object by defining each message that the object can receive and the related operation that the object performs.

(ii) **An Implementation description:** This description shows implementation details for each operation implied by a message that is passed to an object.

- Procedural details that describe operations.
- Internal details about the data structures that describe the object's attributes.
- Information about the object's private part.

- Object design process involves steps as shown:

Step 1 : Obtain operations for the object from the other models.
Step 2 : Choose a minimum cost algorithm with a proper database.
Step 3 : Choose an optimum path which takes minimum cost.
Step 4 : Implement software control.
Step 5 : Use proper class structure and use inheritance between them.
Step 6 : Design implementation of association.
Step 7 : Determine the exact representation of object attributes.
Step 8 : Package classes and association into modules.

2. **Designing Algorithms and Data Structures:**

- A variety of representations contained in the analysis model and the system design provide a specification for all operations and attributes.
- Algorithms and data structures are designed using an approach that differs little from the data design and component-level design approaches discussed for conventional software engineering.

(i) **Algorithm:** An algorithm is created to implement the specification for each operation. In many cases, the algorithm is a simple computational or procedural sequence that can be implemented as a self-contained software module. However, if the specification of the operation is complex, it may be necessary to modularize the operation. Conventional component-level design techniques can be used to accomplish this.

(ii) **Data structures:** They are designed concurrently with algorithms. Since operations invariably manipulate the attributes of a class, the design of the data structures that best reflect the attributes will have a strong bearing on the algorithmic design of the corresponding operations.

3. **Program Components and Interfaces:**

- An important aspect of software design quality is modularity; that is, the specification of program components (modules) that are combined to form a complete program.

- The object-oriented approach defines the object as a program component that is itself linked to other components (e.g., private data, operations). But defining objects and operations is not enough.
- During design, we must also identify the interfaces between objects and the overall structure (considered in an architectural sense) of the objects.

Summary

- Object Oriented Design (OOD) involves implementation of the conceptual model produced during object-oriented analysis.
- Object Oriented Methodology (OOM) is a methodology for object oriented development and a graphical notation for representing objects oriented concepts.
- Grady Booch's Object-Oriented Design (OOD), also known as Object-Oriented Analysis and Design (OOAD), is a precursor to the Unified Modeling (UML).
- Yourdon and Coad's Object-Oriented Analysis and Design (OOA/OOD) is an object-oriented method that precedes UML.
- The Coad and Yourdon method for OOD was developed by studying how "effective object-oriented designers" do their design work.
- OOSE was developed by Ivar Jacobson in 1992. OOSE is the first object oriented design methodology that employs use cases in software design.
- Rambaugh method (OMT) describes the analysis, design and implementation of a system.
- The object modeling technique encompasses a design activity that encourages design to be conducted at two different levels of abstraction.
- System design can be defined as, "the process of taking a logical model of a system together with a strongly stated set of objectives/goals for that system and producing the specifications of a physical system that will meet those objectives/goals."
- In OO system design, we partition the analysis model to define cohesive collections of classes, relationships and behavior. These design elements are packaged as a subsystem.
- Data management encompasses two distinct areas of concern:
  - The management of data that are critical to the application itself and
  - The creation of an infrastructure for storage and retrieval of objects.

Check Your Understanding

1. In the Design phase, which is the primary area of concern?
   (a) Architecture (b) Data
   (c) Interface (d) All of the mentioned

2. Which of the following points related to Object-Oriented Development (OOD) is true?
   (a) OOA is concerned with developing an object model of the application domain
   (b) OOD is concerned with developing an object-oriented system model to implement requirements
   (c) All of the mentioned
   (d) None of the mentioned
3. Which of the following is a disadvantage of OOD?
   (a) Easier maintenance
   (b) Objects may be understood as stand-alone entities
   (c) Objects are potentially reusable components
   (d) None of the mentioned
4. How many layers are present in the OO design pyramid?
   (a) three (b) four
   (c) five (d) one
5. Grady Booch, James Rumbaugh, and Ivar Jacobson combined the best features of their individual object-oriented analysis into a new method for object oriented design known as _____.
   (a) HTML (b) XML
   (c) UML (d) SGML
6. What kind of approach was introduced for elicitation and modelling to give a functional view of the system?
   (a) Object Oriented Design (by Booch)
   (b) Use Cases (by Jacobson)
   (c) Fusion (by Coleman)
   (d) Object Modeling Technique (by Rumbaugh)
7. Object modeling technique consists of which of following phase/s ?
   (a) Analysis (b) System design
   (c) Object design (d) Implementation
   (e) All of the above
8. OMT separates modeling into which of the following part/s?
   (a) Object model (b) Functional model
   (c) Dynamic model (d) All of the above
9. Booch methodology covers _____ and _____ phases of the object-oriented system.
   (a) analysis; design (b) design;coding
   (c) analysis; development (d) None of the above

10. Booch methodology is criticized for its _____.
    (a) Analysis phase (b) Design phase
    (c) Development phase (d) A large set of symbols
11. Which of the following steps are covered by the micro development process of Booch?
    (a) Identify classes and objects.
    (b) Identify Class and object semantics.
    (c) Identify class and object relationships.
    (d) Identify class and object interface and their implementations.
    (e) All of the above

Answers

1. (d)	2. (c)	3. (d)	4. (b)	5. (c)	6. (b)	7. (e)	8. (d)	9. (a)	10. (d)	11. (e)

Practice Questions

Q.I Answer the following questions in short.

1. What is meant by object oriented design?
2. Enlist various design issues of object oriented design.
3. Enlist various generic components of object oriented design models.
4. How to partition an analysis model?
5. What is meant by concurrency?
6. Define data management component.

Q.II Answer the following questions.

1. With a suitable diagram, describe the system design process.
2. Describe the following object oriented methods:
   (a) Booch
   (b) Coad and Yourdon
   (c) Jacobson
   (d) Rambaugh
3. Describe various task management components.
4. Describe term: Inter subsystem communication.
5. Explain the object design process in detail.
6. Explain concurrency and subsystem allocation.
7. Describe the following models:
   (a) Object Model
   (b) Dynamic Model

Q.III Define the terms.

1. Notation
2. Micro Process
3. Dynamic model
4. Sub system
5. Physical model

Previous Exam Questions

Summer 2018

1. Define task management component.
Ans. Refer to Section 9.4.3. [2 M]
2. Describe the Jacobson method in details.
Ans. Refer to Section 9.2.3. [4 M]

Winter 2018

1. Explain generic components of the object oriented design model.
Ans. Refer to Section 9.3. [4 M]
2. Explain System Design Process.
Ans. Refer to Section 9.4. [4 M]

Summer 2019

1. What is meant by concurrency?
Ans. Refer to Section 9.4.2. [2 M]
2. Explain Generic components of the object oriented design model.
Ans. Refer to Section 9.3. [4 M]
3. Describe the Rambaugh method in details.
Ans. Refer Section 9.2.4. [4 M]
4. Explain the system design process.
Ans. Refer to Section 9.4. [4 M]

■■■

Bibliography

Books:

1. Strategic Enterprise Architecture Management: Challenges, Frederik Ahlemann, Eric Stettiner, Marcus Messerschmidt.

Websites:

1. https://www.projectmanager.com
2. https://www.wrike.com/project-management-guide

■■■