Unit II - REQUIREMENT ANALYSIS AND SPECIFICATION

Requirement analysis and specification – Requirements gathering and analysis – Software Requirement Specification – Formal system specification – Finite State Machines – Petrinets – Object modelling using UML – Use case Model – Class diagrams – Interaction diagrams – Activity diagrams – State chart diagrams – Functional modelling – Data Flow Diagram- CASE TOOLS.

Requirement Analysis and Gathering

Requirement Analysis

✓ Requirement Analysis is an important phase of SDLC.

✓ A reasonable product delivery demands correct requirements gathering, the efficient examination of requirements gathered, and clear requirement documentation as a precondition.

✓ This entire process is known as **Requirement Analysis in Software Development Life Cycle.**

Requirements Analysis Process

The Requirement Analysis phase is SDLC's first activity, followed by Functional Specifications and so on.

Requirement Analysis begins with:

1. ***Requirement Gathering***, which is also known as ***Elicitation.***
2. It is followed by ***analyzing*** the collected requirements to understand the feasibility and correctness of converting the requirements into a possible product.
3. Then, ***Documenting*** the gathered requirements.

1) Requirement Gathering

✓ To make sure that all the steps mentioned above are appropriately executed, clear, concise, and correct requirements must be gathered from the customer.

✓ The customer should be able to define their requirements properly and the business analyst should be able to collect them in the same way the customer is intending it to convey them.

2) Analyzing Gathered Requirements

✓ Post requirement gathering, analysis of requirement starts. At this stage, various stakeholders sit and do a brainstorming session.

✓ They analyze the requirements gathered and look for the feasibility to implement them. They discuss it with each other and any ambiguity is sorted out.

- ✓ *This step is important in the requirement analysis process because*
  - ➢ Customer may provide some requirements which could be impossible to implement due to various dependencies.
  - ➢ A business analyst might have understood the requirement from the ***customer*** differently than how a ***programmer*** would have interpreted it.

3) Documenting Analyzed Requirements

- ✓ Once the analysis of requirements is done requirement documentation starts.
- ✓ Various types of requirements come out of the analysis of requirements.

Some of these are:

- ➢ Customer Requirement specification.
- ➢ Software Architecture requirement.
- ➢ Software Design requirement.
- ➢ Functional Requirement specification (directly derived from customer specifications.)
- ➢ Non-functional Requirement specification (viz. performance, stress, load, etc.).
- ➢ User Interface requirements.

Goals of Requirements Analysis

- ✓ Requirements analysis or requirements engineering is a process used to determine the needs and expectations of a new product.
- ✓ It involves frequent communication with the stakeholders and end-users of the product to define expectations, resolve conflicts, and document all the key requirements.

Requirements Specification (SRS) Document

- ✓ A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform.
- ✓ It also describes the functionality the product needs to fulfil the needs of all stakeholders (business, users).
- ✓ A good SRS document will define everything from how software will interact when embedded in hardware to the expectations when connected to other software.

The elements that comprise an SRS summarized into four Ds:

- ***Define*** *your product's purpose.*
- ***Describe*** *what you're building.*
- ***Detail*** *the requirements.*
- ***Deliver*** *it for approval.*

Why Use an SRS Document?

- ✓ An SRS gives a complete picture of your entire project. It provides a single source about every team involved in development will follow.
- ✓ It is a plan of action and keeps all the teams — from development and testing to maintenance — on the same page.

Software Requirements Specification vs System Requirements Specification

- ✓ A **system requirements specification (SyRS)** presents general information on the requirements of a system, which may include both hardware and software, based on an analysis of business needs.
- ✓ A **software requirements specification (SRS)** details the specific requirements of the software that is to be developed.

To Write an SRS Document

Creating a clear and effective SRS document can be difficult and time-consuming. But it is critical to the efficient development of a high-quality product that meets the needs of business users.

The five steps to write an effective SRS document

1. Define the Purpose with an Outline (r use an SRS Template)

- ✓ The first step is to create an outline for a software requirements specification. This may be something a user can create themselves, or use an existing SRS template.
- ✓ For creating the outline, it might include:

Introduction

- ✓ Purpose
- ✓ Product Scope
- ✓ Definitions and Acronyms

Overall Description

- ✓ User Needs
- ✓ Assumptions and Dependencies

System Features and Requirements

- ✓ Functional Requirements
- ✓ External Interface Requirements
- ✓ System Features
- ✓ Nonfunctional Requirements

2. Define your Product's Purpose

Product Scope

- ✓ What are the benefits, objectives, and goals we intend to have for this product?
- ✓ This should relate to overall business goals, especially if teams outside of development will have access to the SRS.
- ✓ Clearly define all key terms, acronyms, and abbreviations used in the SRS.
- ✓ This will help eliminate any ambiguity and ensure that all parties can easily understand the document.

3. Describe What You Will Build

- ✓ It is to give a description of what you're going to build. Why is this product needed? Who is it for? Is it a new product? Describe who will use the product and how.
- ✓ Who will be using the product? Are they a primary or secondary user?
- ✓ What is their role within their organization? What need does the product need to fulfil for them?

4. Detail Your Specific Requirements

It includes functional requirements, interface requirements, system features, and various types of nonfunctional requirements:

a. Functional Requirements - essential to the product because, it provide some sort of functionality to the software product.

- ***External Interface Requirements –*** a specific types of functional requirements. They outline how the product will interface with other components such as User, Hardware, Software and their Communications.
- ***System Features -*** which are required for a system to function.

b. Non-functional Requirements - which help ensure that a product will work the way users and other stakeholders expect it to, can be just as important as functional ones.

- ✓ These may include:
  - ➢ Performance requirements
  - ➢ Safety requirements
  - ➢ Security requirements
  - ➢ Usability requirements
  - ➢ Scalability requirements

5. Deliver for Approval

- ✓ After completing the SRS, it needs to get it approved by key stakeholders.
- ✓ This will require everyone to review the latest version of the document.

Unified Modeling Language (UML)

- ✓ UML can be called *a graphical modeling language* that is used in the field of software engineering. It specifies, visualizes, builds, and documents the software system's artifacts (main elements). It *is a visual language*, not a programming language.
- ✓ UML diagrams are used *to depict the behavior and structure of a system.*
- ✓ UML facilitates modeling, design, and analysis for software developers, business owners, and system architects.

Characteristics of UML

- ✓ It is a modeling language that has been generalized for various use cases.
- ✓ It is not a programming language; instead, it is a graphical modeling language that uses diagrams that can be understood by non-programmers as well.
- ✓ It has a close connection to object-oriented analysis and design.
- ✓ It is used to visualize the system's workflow.

The Benefits of Using UML

- ✓ Because it is a general-purpose modeling language, it can be used by any modeler. UML is a straightforward modeling approach utilized to describe all practical systems.
- ✓ Because no standard methods were available then, UML was developed to systemize and condense object-oriented programming.

- ✓ The UML diagrams are designed for customers, developers, laypeople, or anyone who wants to understand a system, whether software or non-software. Businesspeople do not understand code.
- ✓ As a result, UML becomes vital for communicating the system's essential requirements, features, and procedures to non-programmers. Technical details became easier to understand by non-technical people through an introduction to UML.
- ✓ When teams can visualize processes, user interactions, and the system's static structure, it saves a lot of time in the long run.

Types of UML Diagrams

Three classifications of UML diagrams

- ✓ **Behavior diagrams -** A type of diagram *that depicts behavioral features of a system or business process*. This includes ***activity, state machine, and use case diagrams as well as interaction diagrams.***
- ✓ **Interaction diagrams -** *A subset of behavior diagrams which emphasize object interactions*. This includes ***communication, interaction overview, sequence, and timing diagrams.***
- ✓ **Structure diagrams -** A type of diagram that *depicts the elements of a specification that are irrespective of time.* This includes ***class, composite structure, component, deployment, object, and package diagrams***.

Use Case Model

In the Unified Modeling Language (UML), a use case diagram can summarize the details of the system's users (also known as actors) and their interactions with the system. A set of specialized symbols and connectors were used.

An effective use case diagram can help your team discuss and represent:

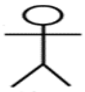
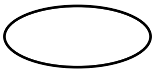
- ✓ Scenarios in which the system or application interacts with people, organizations, or external systems
- ✓ Goals that the system or application helps those entities (known as actors) achieve
- ✓ The scope of the system

Use Case Diagram

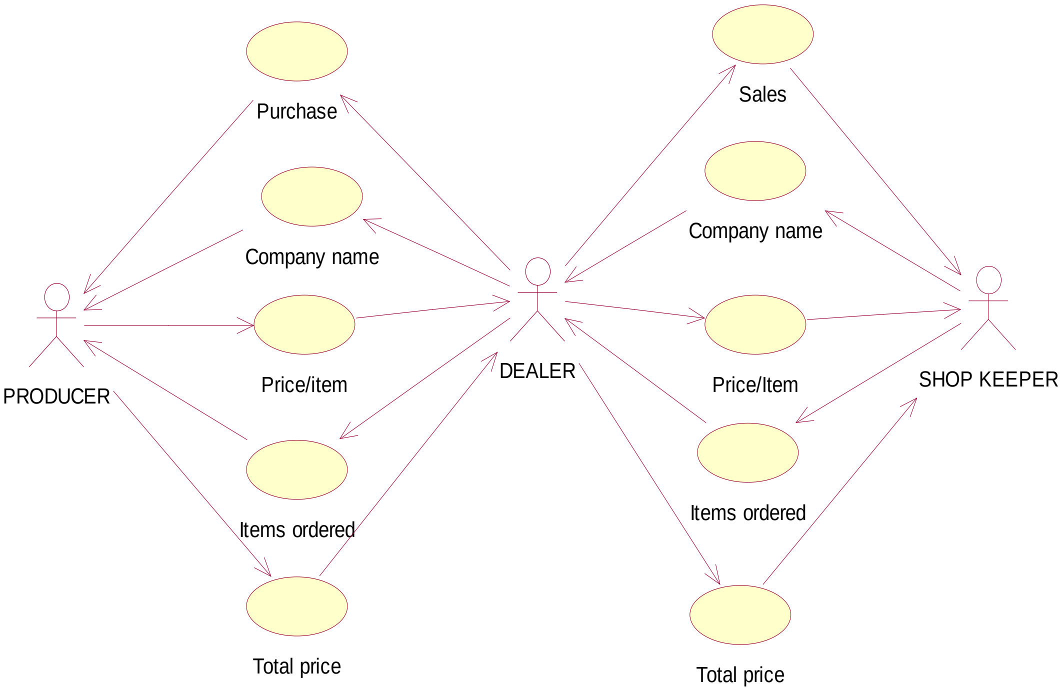
Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors.

The use cases and actors in use-case diagrams describe what the system does and how the actors use it, but not how the system operates internally.

Symbolic Notations:

Symbols	**Name**	**Description**
	Actors	Actor may be a human, system and organisation. Each actor has a unique name that describes the role of the user who interacts with the system.
	Use Case	A use case describes a function that a system performs to achieve the user's goal. Each use case must have a unique name that describes the action that the system performs.
⟶	Association	Association used to represent the relationship between use case and the actors
——▷	Generalization	A generalization relationship is a relationship in which one model element (the child) is based on another model element (the parent).
- - - - - - - -► <<include>> - - - - - - - -► <<extend>> - - - - - - - -►	Dependency include extend	A dependency relationship is a relationship in which one element or client depends on another element. In dependency, "include" or "extends concepts are labelled." A "Include" relationship *indicates that one use case is needed by another* in order to perform a task. An "extends" relationship *indicates alternative options* under a certain use case.

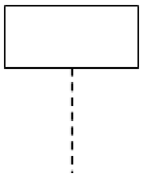
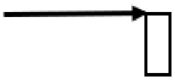
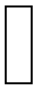
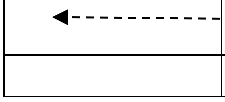
Example of Use Case Diagram - Stock Maintenance System



Sequence Diagram

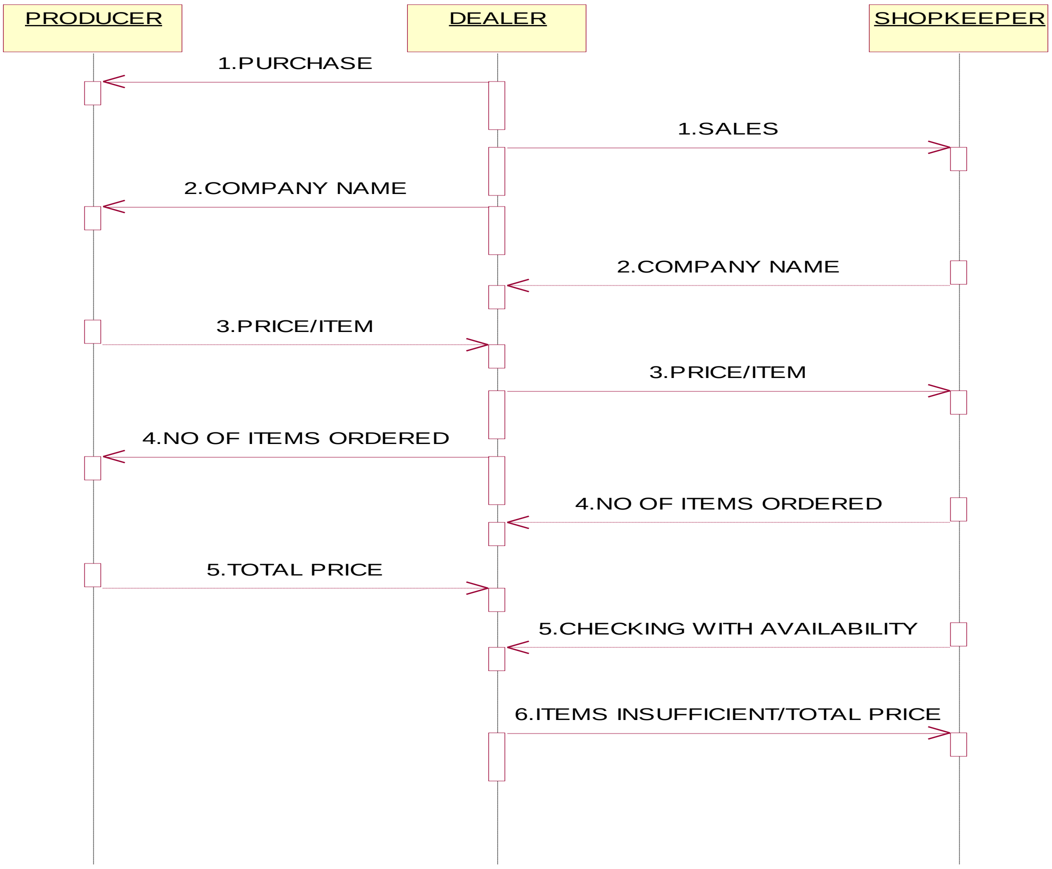
The sequence diagram is used primarily to show the interactions between objects in the sequential order (i.e.) a step-by-step activity.

Symbolic Notations:

Symbols	Name	Description
	Object Lifeline	Lifeline notation is placed across the top of the diagram. Used to start the conversation between the objects. The lifeline's name is placed inside the box.
	Request Message	Used to represent the message convey from one to another objects.
	Activation or Synchronization Bar	used *to indicate that an object is active during an interaction between two objects*.
	Response Message	Used to get the response from object.
	Message to self	Used to convey the message within the objects.

↵		
✕	Destruction	Used to indicate the stop of conversation within the objects and saves the object life.

Example for Sequence Diagram



State Chart Diagram

Statecharts define the run-time behavior of instances of a class. A statechart is used to represent the abstract representation of use case or objects. A message triggers a transition from one state to another.

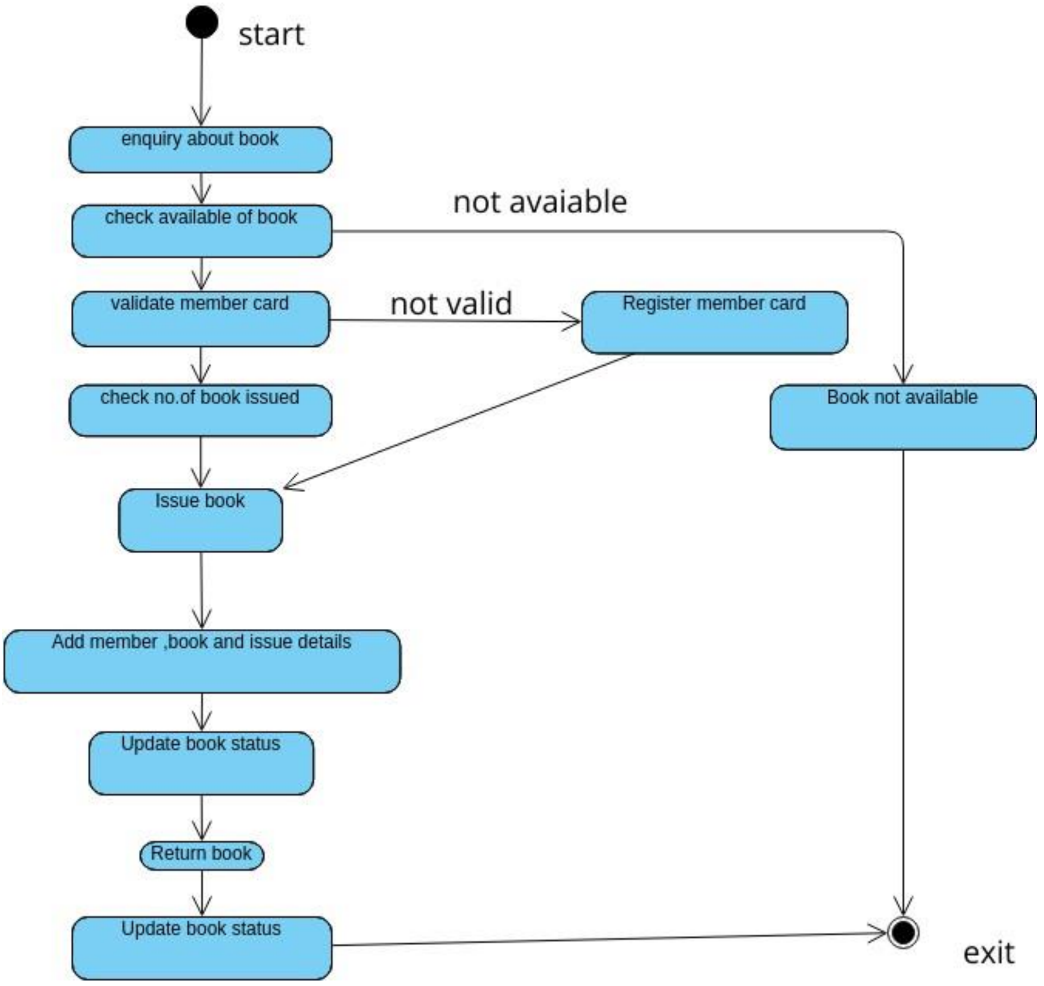
A state machine is any device that stores the status of an object at a given time and can change status or cause other actions based on the input it receives.

Symbolic Notations:

Symbols	**Name**	**Description**
●	Initial / Start state	Used to indicate the start of process.

◉	Final / End State	Used to indicate the end of process.
▭	State/Activity	Represents the process or action of the user. States represent situations during the life of an object.
→	Transition	Represents the path between different states of an object. Used to move from one state to another.

Example for State Chart – Library Management System

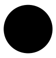
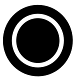
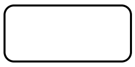
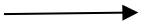
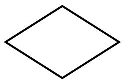
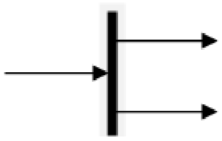
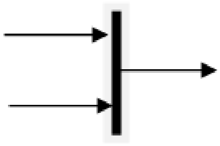
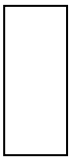


Activity Diagram

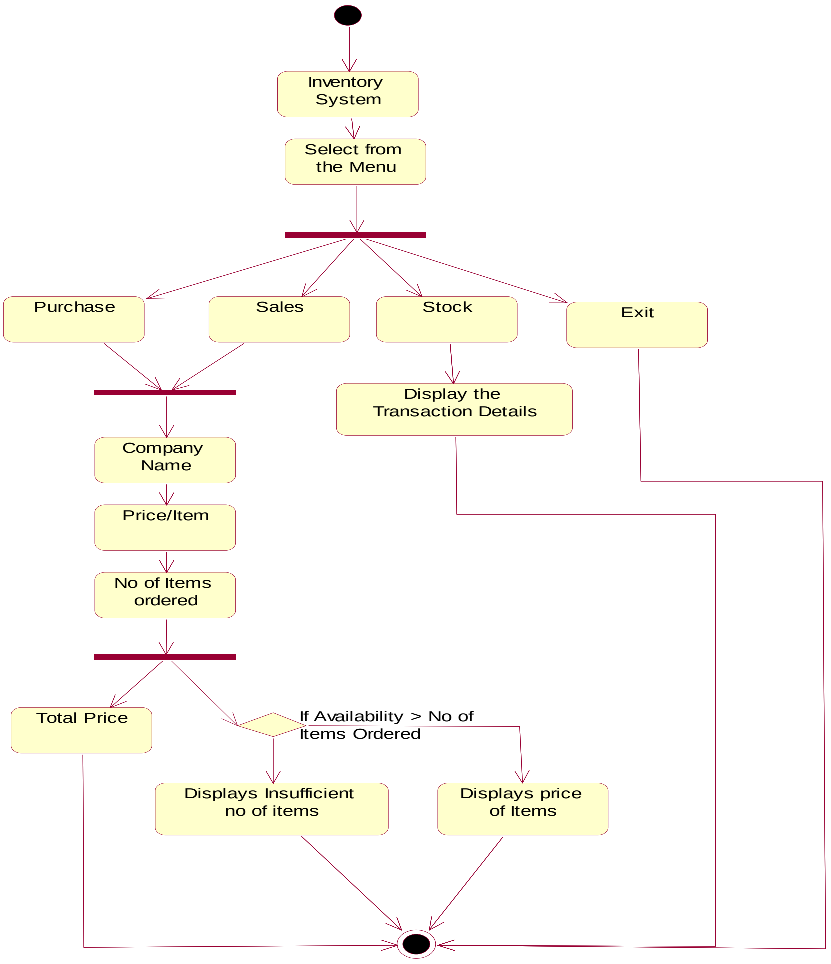
In UML, an activity diagram provides a view of the behavior of a system by describing the sequence of actions in a process. Activity diagrams are similar to flowcharts because they show the flow between the actions in an activity.

Activity diagrams can also show parallel or concurrent flows and alternate flows.

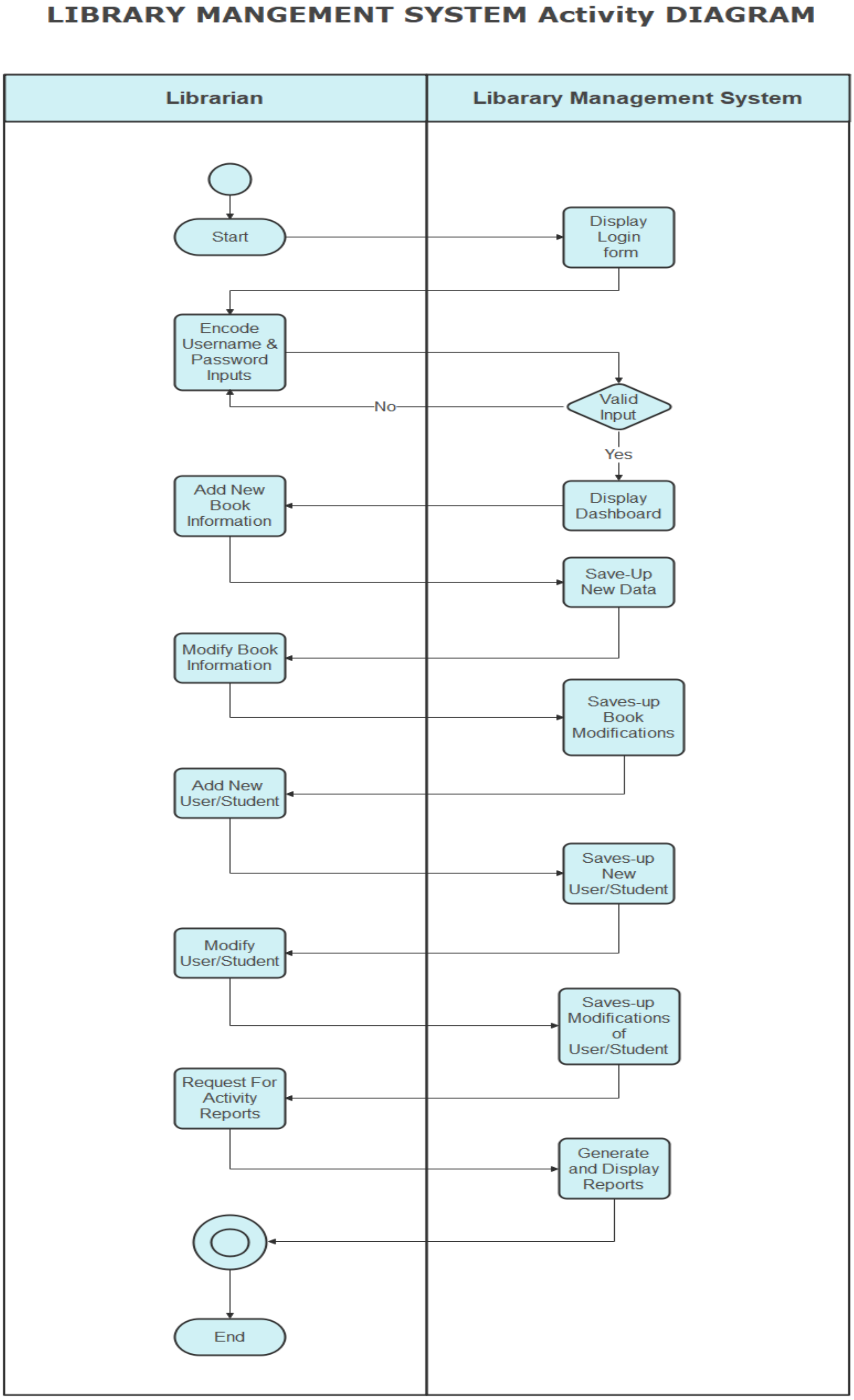
Symbolic Notations:

Symbols	Name	Description
	Initial / Start state	Used to indicate the start of process.
	Final / End State	Used to indicate the end of process.
	State/Activity	Represents the process or action of the user. States represent situations during the life of an object.
	Transition	Represents the path between different states of an object. Used to move from one state to another.
	Decision	Used to represent a conditional branch point with one input and multiple outputs
	Fork	A Fork node is a control node that splits a flow into multiple concurrent flows. This will have one incoming edge and multiple outgoing edges.
	Join	A join node is a control node that synchronizes multiple flows. This will have multiple incoming edges and one outgoing edge.
	Swimlame (also known as partitions)	A boundaries or borderlines which separate an organization's activities from the activities of other organizations. The **swimlane activity diagram** clearly states who does what in a process. Swimlanes must be arranged in a logical order.

Example for Activity Diagram – Stock Maintenance System



Activity Diagram using Swimlane:

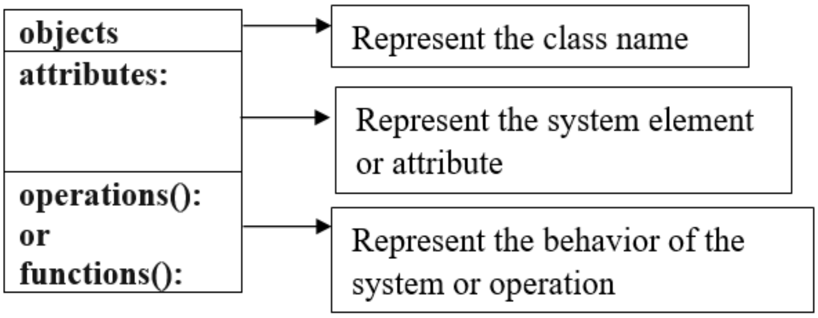

LIBRARY MANGEMENT SYSTEM Activity DIAGRAM
Librarian
Libarary Management System
Start
Display Login form
Encode Username & Password Inputs
Valid Input
No
Yes
Display Dashboard
Add New Book Information
Save-Up New Data
Modify Book Information
Saves-up Book Modifications
Add New User/Student
Saves-up New User/Student
Modify User/Student
Saves-up Modifications of User/Student
Request For Activity Reports
Generate and Display Reports
End

Class diagrams

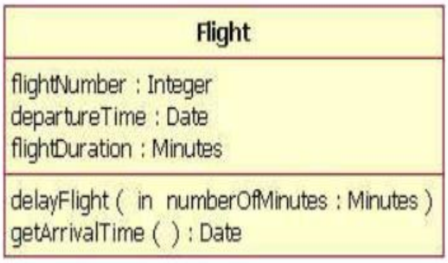
- ✓ Class diagrams are fundamental to the object modeling process and model the static structure of a system.
- ✓ Class diagrams are the blueprints of the system or subsystem.
- ✓ Class diagrams are used
  - to model the objects that make up the system,
  - to display the relationships between the objects, and
  - to describe what those objects do and the services that they provide.

Notations:

The UML representation of a class is a rectangle containing three compartments stacked vertically.


objects
attributes:
operations():
or
functions():
Represent the class name
Represent the system element or attribute
Represent the behavior of the system or operation

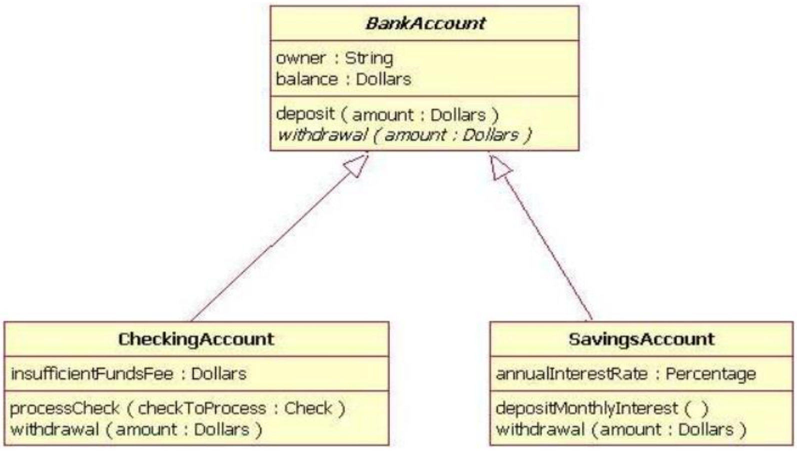
Example Of Class Diagram



Elements used in Class Diagrams:

- ✓ **Classes -** a *class* represents an object or a set of objects that share a common structure and behavior.
- ✓ **Objects -** *objects* are model elements that represent instances of a class or of classes.
- ✓ **Signals -** *signals* are model elements that are independent of the classifiers that handle them. Signals specify one-way, asynchronous communications between active objects.
- ✓ **Data types -** *data types* are model elements that define data values. It represents primitive types, such as integer or string types, and enumerations, such as user-defined data types.
- ✓ **Artifacts -** *artifacts* are model elements that represent the physical entities in a software system. Artifacts represent physical implementation units, such as executable files, libraries, software components, documents, and databases.
- ✓ **Relationships in class diagrams -** a relationship is a connection between model elements.

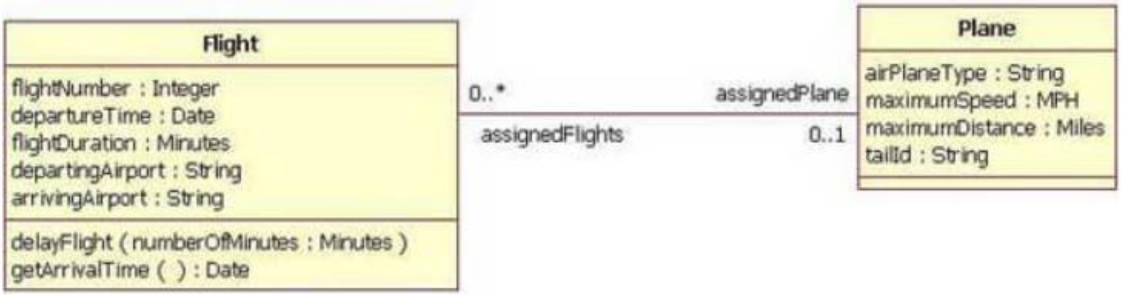
- ✓ **Inheritance -** refers to the ability of one class (child class) to *inherit* the identical functionality of another class (super class), and then add new functionality of its own.


BankAccount
owner : String
balance : Dollars
deposit (amount : Dollars)
withdrawal (amount : Dollars)
CheckingAccount
insufficientFundsFee : Dollars
processCheck (checkToProcess : Check)
withdrawal (amount : Dollars)
SavingsAccount
annualInterestRate : Percentage
depositMonthlyInterest ( )
withdrawal (amount : Dollars)

- ✓ **Associations -** In a system, certain objects will be related to each other, and these relationships themselves need to be modelled for clarity.

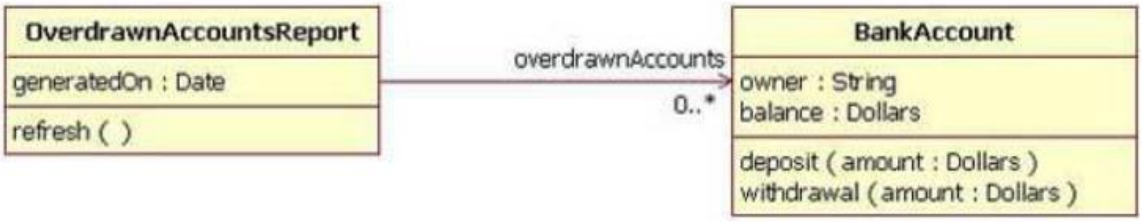
1. **Bi-directional (standard) Association**

- ✓ A bi-directional association is indicated by a solid line between the two classes.
- ✓ An association is a linkage between two classes. Associations are always assumed to be bi-directional; this means that both classes are aware of each other and their relationship.



2. **Uni-directional association**

- ✓ A uni-directional association is drawn as a solid line with an open arrowhead (not the closed arrowhead, or triangle, used to indicate inheritance) pointing to the known class.
- ✓ In a uni-directional association, two classes are related, but only one class knows that the relationship exists.

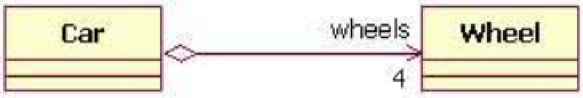


✓ **Aggregation -** Aggregation is a special type of association used to model a "whole to its parts" relationship. In basic aggregation relationships, the lifecycle of a *part* class is independent from the *whole* class's lifecycle.

1. **Basic Aggregation**

✓ An association with an aggregation relationship indicates that one class is a part of another class. In an aggregation relationship, the child class instance can outlive its parent class.

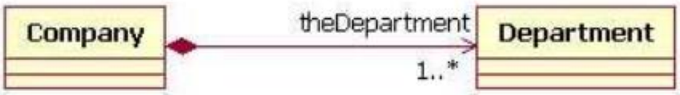
✓ To represent an aggregation relationship, you draw a solid line from the parent class to the part class, and draw an unfilled diamond shape on the parent class's association end.



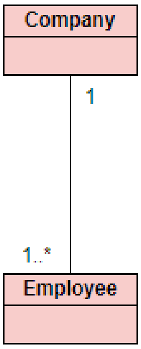
2. **Composition Aggregation**

✓ The composition aggregation relationship is just another form of the aggregation relationship, but the child class's instance lifecycle is dependent on the parent class's instance lifecycle.

✓ The composition relationship is drawn like the aggregation relationship, but this time the diamond shape is filled.



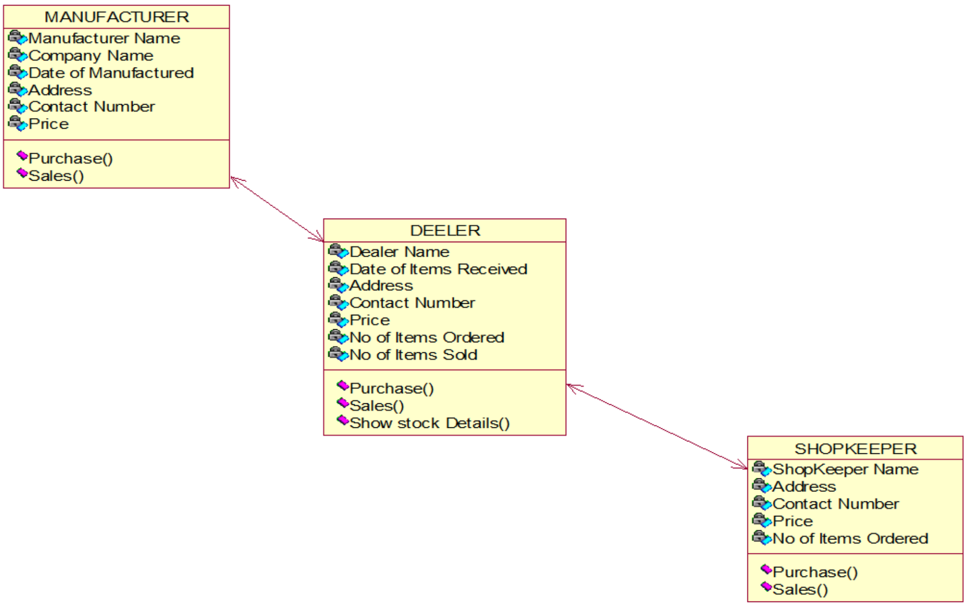
✓ **Multiplicity -** Place multiplicity notations near the ends of an association. These symbols indicate the number of instances of one class linked to one instance of the other class. For example, one company will have one or more employees, but each employee works for one company only.



Multiplicities examples:

1	Exactly one, no more and no less
0..1	Zero or one
*	Many
0..*	Zero or many
1..*	One or many

Example for Class Diagram – Stock Maintenance System



Formal system specification

A formal software specification is a statement expressed in a language whose vocabulary, syntax, and semantics are formally defined. The need for a formal semantic definition means that the specification languages cannot be based on natural language; it must be based on mathematics.

Formal specifications are mathematically based techniques whose purpose are to help with the implementation of systems and software.

A good specification will have:

- Constructability, manageability and evolvability
- Usability
- Communicability
- Powerful and efficient analysis

In formal specifications, it provide an ability to perform proofs on software implementations. These proofs may be used to validate a specification, verify correctness of design, or to prove that a program satisfies a specification.

These types of models can be categorized into the following specification paradigms:

- History-based specification
- ***State-based specification – Finite State Machine***
- ***Transition-based specification - Petrinets***
- Functional specification
- Operational Specification

State Machine Diagrams

State machine diagram typically is used to describe state-dependent behavior for an object. **An object responds differently to the same event depending on what state it is in**.

State machine diagrams are usually applied to objects but can be applied to any element that has behavior to other entities such as: actors, use cases, methods, subsystems systems and etc. and they are typically used in conjunction with interaction diagrams (usually sequence diagrams).

Characteristics of State

- State represent the conditions of objects at certain points in time.
- Objects (or Systems) can be viewed as moving from state to state
- A point in the lifecycle of a model element that satisfies some condition, where some particular action is being performed or where some event is waited

The basic building blocks of a state machine are states and transitions. In State Machine Model,

- ✓ ***State*** – It represents the *current situation.* One state is marked as ***the initial state -*** where the execution of the machine starts. And ***the final state*** of a state machine diagram is shown as concentric circles. An open loop state machine represents an object that may terminate before the system terminates.
- ✓ ***Event*** – It represents *the input to the state.*
- ✓ ***Action*** – It represents *the activity performed* by the state machine.
- ✓ ***Transition*** – It represents the *movement from one state to another*.

In State Machine, the object can be classified into

1. **State-Independent Object -** If an ***object always responds the same way to an event***, then it is considered state-independent with respect to that event. The object is state-independent with respect to that message.
2. **State-Dependent Objects** – If ***the objects react differently to events*** depending on their state or mode.

Finite State Machine (FSM)

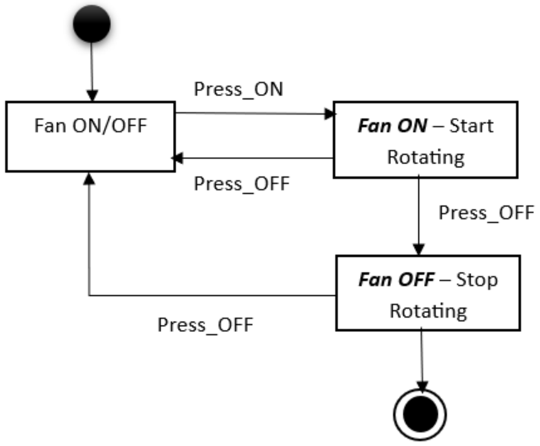
A state machine ***is a behavior model***. It consists of ***a finite number of states*** and is therefore also called ***finite-state machine (FSM).*** Based on the current state and a given input the machine performs state transitions and produces outputs. There are basic types like Mealy and Moore machines.

Types of FSM

There are two basic types of finite-state machines (FSM)

Moore Machines

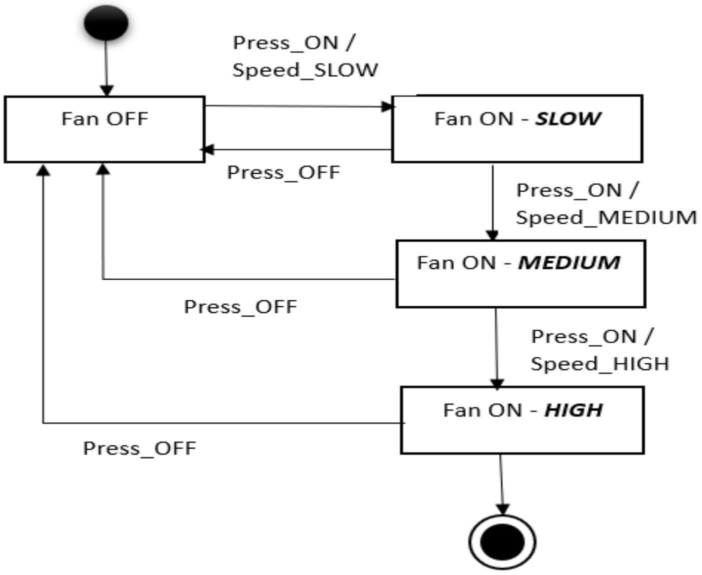
- ✓ Moore machines consist of states and transitions.
- ✓ Moore machines produce *outputs* - ***based on current state, not by any input.***



The states (OUTPUT) here is : ON and OFF

Mealy Machines

- ✓ Mealy machines consist of states and transitions.
- ✓ Mealy machines produce outputs **-** ***based on transitions,*** *not in states.*
- ✓ This often results in state diagrams with fewer states because more logic can be put on transitions.



The transitions (OUTPUT) here is : SLOW, MEDIUM and HIGH

Difference between Moore and Mealy Machines

Moore Machine	**Mealy Machines**
1. Output depends only upon the present state.	1. Output depends on the present state as well as present input.
2. Output is placed on states.	2. Output is placed on transitions.
3. More states are required.	3. Less number of states are required.
4. If input changes, output does not change.	4. If input changes, output also changes
5. There is less hardware requirement for circuit implementation.	5. There is more hardware requirement for circuit implementation.
6. They react slower to inputs.	6. They react faster to inputs.
7. Synchronous output and state generation.	7. Asynchronous output generation.
8. Easy to design.	8. It is difficult to design.

Petri-nets

Petri nets — Formal technique for describing concurrent interrelated activities

✓ Petri nets are a graphical for representing a system in which there are multiple independent

✓ activities in progress at the same time. The ability to model multiple activities differentiates Petri nets from finite state machines.

✓ In a finite state machinethere is always a single “current” state that determines which

✓ action can next occur.

✓ In Petri nets there may be several states any one of which may evolve by changing the state of the Petri net.

A Petri net consists of four elements: places, transitions, edges, and tokens.

✓ *Graphically, Places are represented by circles* ○

✓ *Transitions by rectangles* □ *or* |

✓ *Edges by directed arrow* →

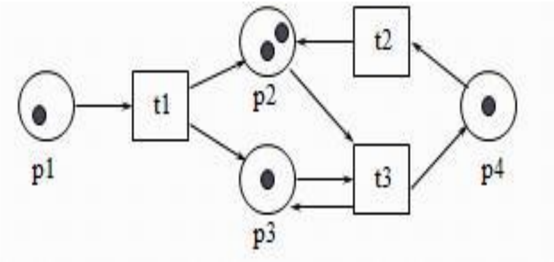
✓ *Tokens by small solid (filled) circles* ●

The Classical Petri Net Model:

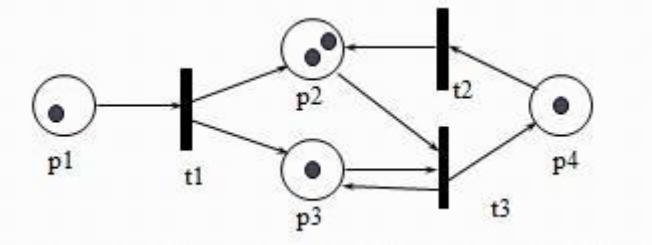
✓ A Petri net is a network composed of **places(O) and transitions(Π)**

✓ **Connections** are directed and between a place and a transition.

✓ Tokens are the dynamic objects.



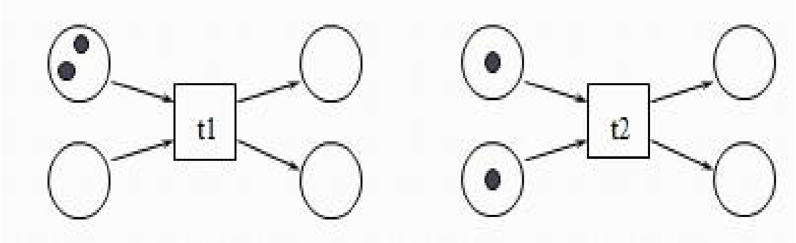
Another notation is to use a solid bar for the transitions.



- ✓ The **state** of a Petri net is determined by the distribution of tokens over the places (we could represent the above **state** as (1,2,1,1) for (p1, p2, p3, p4))
- ✓ Transition t1 has three **input places** (p1, p2 and p3) and two **output places** (p3 and p4).
- ✓ Place p3 is both an input and an output place of t1.

Enabling Condition:

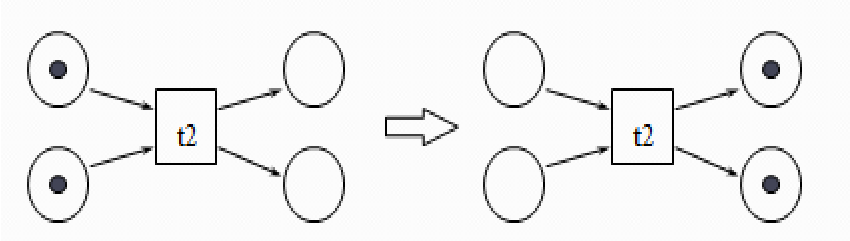
- ✓ Transitions are the **active** components and places and tokens are **passive** components.
- ✓ A transition is **enabled** if each of the input places contains tokens.



Transition t1 is not enabled, transition t2 is enabled.

Firing:

- ✓ An enabled transition may fire.
- ✓ Firing corresponds to consuming tokens from the input places and producing tokens for the output places.



Firing is atomic (only one transition fires at a time, even if more than one is enabled)

Functional Modeling

- ✓ **Functional Modelling** provides the outline that what the system is supposed to do.
- ✓ It does not describes what is the need of evaluation of data, when they are evaluated and how they are evaluated apart from all it only represent origin of data values.
- ✓ It describes the function of internal processes with the help of DFD.

Data Flow Diagram

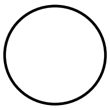
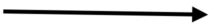
- ✓ Function modelling is represented with the help of DFDs.
- ✓ A **data flow diagram** is a visual representation of the flow of information for a process or a system.
- ✓ A data flow diagram (DFD) maps out the sequence of information, actors, and steps within a process or system. It uses a set of defined symbols that each represent the people and processes needed to correctly transmit data within a system.
- ✓ A DFD can be as simple or as complex as the system it represents, but the easiest way to make one is with a Data Flow Diagram tool.

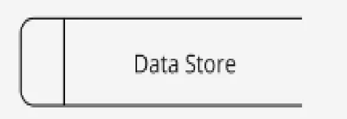
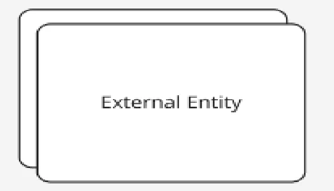
Rules for Making Data Flow Diagrams

- ✓ Every process must have meaningful name and number.
- ✓ Level 0 DFD must have only one process.
- ✓ Every data flow and arrow has given the name.
- ✓ DFD should be logical consistent.
- ✓ DFD should be organised in such a way that it is easy to understand.
- ✓ There should be no loop in the DFD.
- ✓ Each DFD should not have more than 6 processes.
- ✓ The process can only connected with process, external entity and data store.
- ✓ External entity cannot be directly connected with external entity.
- ✓ The direction of DFD is left to right and top to bottom representation.

Components of Data Flow Diagrams

The data flow diagram shows data inputs, outputs, storage sites, and paths between each destination using predetermined symbols and shapes like rectangles, circles, and arrows.

Process 	It is also called function symbol and used to process all the information. If there are calculations so all are done in the process part. It is represented with circle and name of the process and level of DFD written inside it.
Data Flow 	The data passes from one place to another is shown by data flow. Data flow is represented with arrow and some information written over it.

Data Store  	It is used to store the information and retrieve the stored information. A single element or a set of elements can be found in the data storage. The group of parallel lines denotes a location to collect the data items.
Entity  	External entity is the entity that takes information and gives information to the system. It is represented with rectangle.

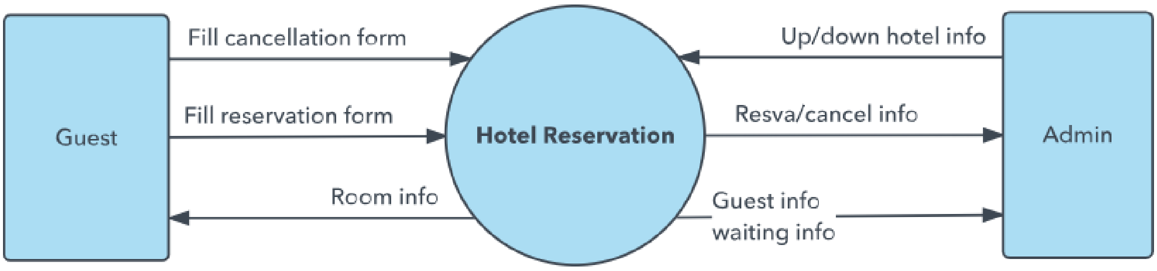
Levels of Data Flow Diagrams

The levels of DFD are Classified into

- ✓ Level 0 DFD
- ✓ Level 1 DFD
- ✓ Level 2 DFD

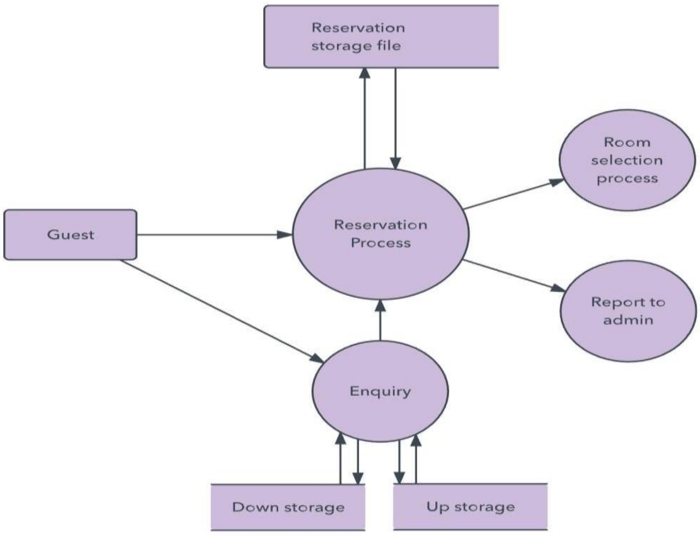
Level 0 DFDs

- ✓ The level 0 data flow diagrams are ***the most basic, and they do not provide every little detail of the information or the structure***; instead, it gives a broad view that can easily be understood.
- ✓ These diagrams are straightforward and show single process nodes and connections of those nodes with externalities.



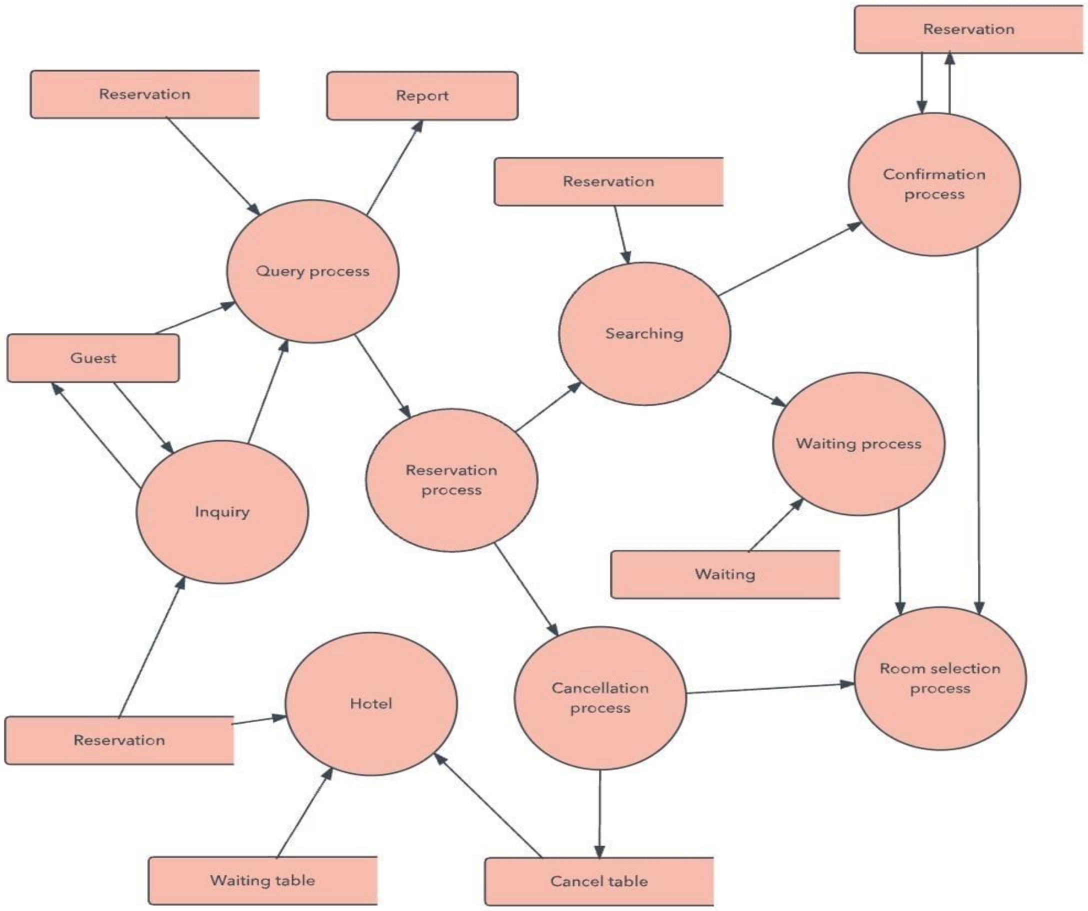
Level 1 DFDs

- ✓ The level 1 data flow diagrams provide more information than the Level 0 DFDs. It ***shows the general overview of the system.***
- ✓ single process node from the level 0 diagram is split into subprocesses in a level 1 data flow diagram.
- ✓ Additional data flows and data stores will be required as these processes are added to the diagram.



Level 2 DFDs

- ✓ The level 2 data flow ***diagrams are way too detailed,*** where processes from Level 1 DFDs are further broken down into more chunks.
- ✓ The objective is to create a map of every little detail of the system to help the engineers to understand and work on it.


Reservation
Reservation
Report
Confirmation process
Reservation
Query process
Searching
Guest
Waiting process
Reservation process
Inquiry
Waiting
Room selection process
Hotel
Cancellation process
Reservation
Waiting table
Cancel table