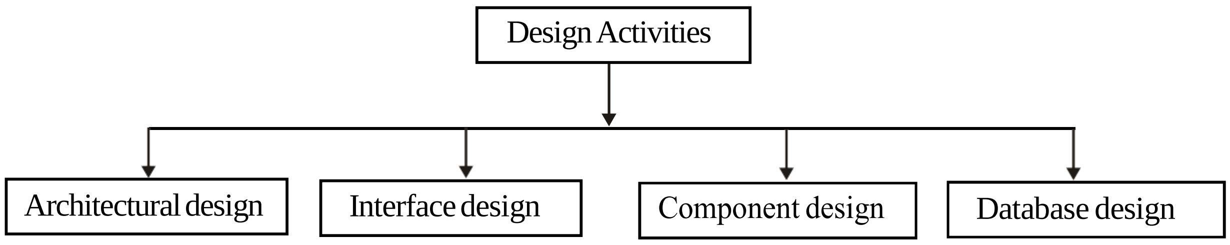
Unit-3
Software Design with UML

❖ DESIGN PROCESS

- The design process is a sequence of steps to describe all aspects of the software.
- Design process specifies how aspect of the system (means how the system will be implemented and how it will work).
- The purpose of the design process is → to plan a solution of problems specified in SRS.
- Design process includes user interface design, input output design, data design, process and program design and technical specification etc.
- It transforming the customer requirements (described in SRS) into appropriate form that is suitable to implement using any of programming languages.
- Output of design process is →design documents.

➢ Classification of design activities:

- Design activities are depending on the type of the software being developed.
- Design activities can be classified like:



Design Activities

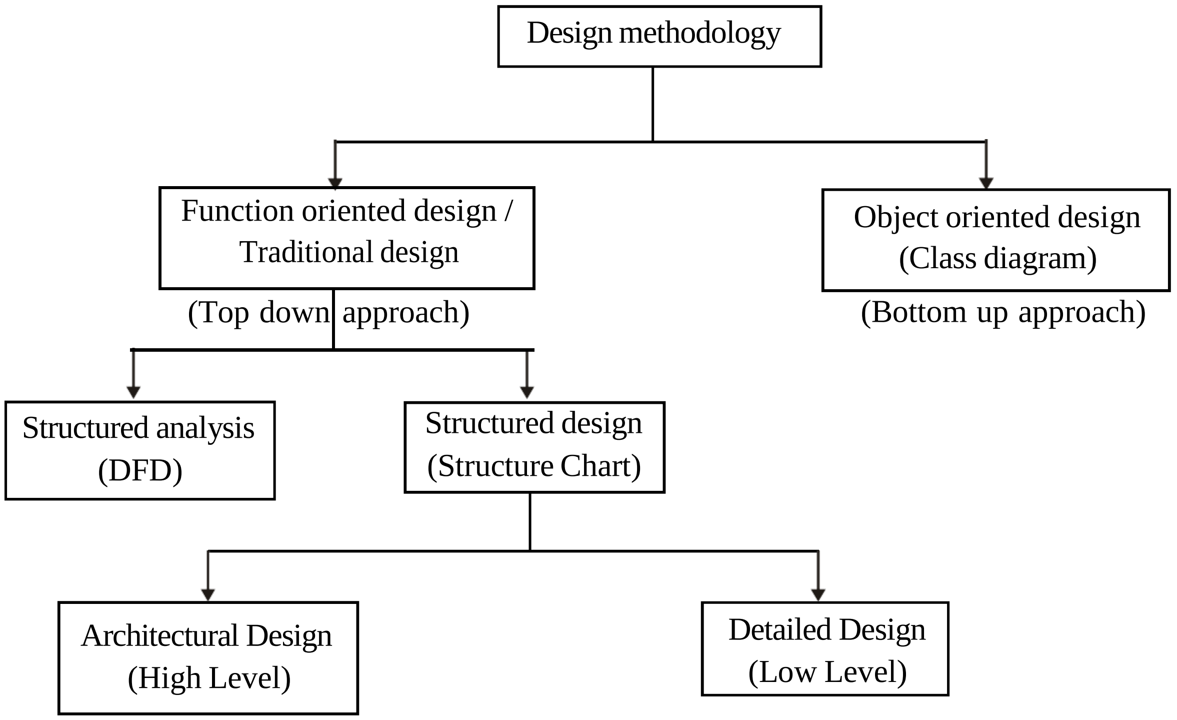
1. **Architectural design:**
   - Where you can identify the overall structure of the system, sub-systems, modules and their relationships.
   - It defines the framework of the computer based system.
   - It can be derived from DFD (data flow diagram).
2. **Interface design:**
   - Where you can define the interface between system components.
   - It describes how system communicates with itself and with the user also.
   - It can be derived from DFD and State transition diagram.
3. **Component design:**
   - That defines each system component and show how they operate.
   - It can be derived from State transition diagram.

4. **Database design:**
   - Where you can define the system data structure and show how they are represented in a database.
   - Existing database can be reused or a new database to be created.
   - The data objects and their relationships are defined in ERD (entity relationship diagram) and detailed content described in data dictionary (DD).
   - It can be derived from ERD and DD.

➢ **Classification of design methodologies:**

- Design methodologies are followed in software development from beginning up to the completion of the product.
- Design methodologies use to provide guidelines for the design activity.
- The nature of the design methodologies are dependent on the following factors:
  - → The software development environment.
  - → The type of the system being developed.
  - → User requirements.
  - → Qualification and training of the development team.
  - → Available software and hardware resources.
- There are large numbers of software design methodologies. Different methodologies are used to solve different type of problems.

Classification of design methodologies is shown in the figure.


Design methodology
Function oriented design / Traditional design
(Top down approach)
Object oriented design (Class diagram)
(Bottom up approach)
Structured analysis (DFD)
Structured design (Structure Chart)
Architectural Design (High Level)
Detailed Design (Low Level)

Design Methodologies

- There are fundamentally two different approaches :

Function oriented design	**Object oriented design**

1. Function oriented design:

- In which the set of functions are described.
- Starting at this high level view of the system, each function is successively refined into more detailed functions or sub functions (top down approach).
- Each of these sub-functions may be split into more detailed sub functions and so on.
- Data in the system is centralized and shared among different functions.
- Function oriented design further classified into $\rightarrow$ Structure analysis and Structure design.

➢ Structure analysis

- It is used to transform a textual description into graphical form.
- It examines the detail structure of the system.
- It identifies the processes and data flow among these processes.
- In structure analysis - functional requirements specified in SRS are decomposed and analysis of data flow is represented here.
- Structure analysis is graphically represented using data flow diagram (DFD).

➢ Structure design

- During structured design, the results of structured analysis are transformed into the software design.
- All the functions identified during analysis are mapped to module structure and that is useful for implementation.
- The aim of the structure design is $\rightarrow$ to transform the result of the structure analysis (DFD) into a structure chart. So structure analysis is graphically represented using structure chart.
- Two main activities are performed during structure analysis :
  - **(i) Architectural design (High level design):** decomposing the system into modules and build relationship among them.
  - **(ii) Detailed design (Low level design):** individual modules are design with data structure and algorithms.
- Problem with the structure design is that $\rightarrow$ if a change is made in one part of the program will require search through of entire program.

2. Object oriented design:

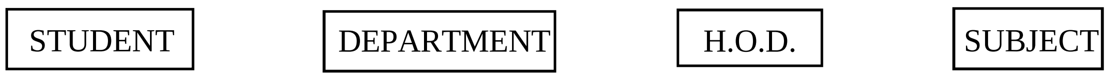
- In this design the system is viewed as a collection of objects (entities).
- Objects available in the systems are identified and their relationships are built during this approach. And the whole system is built (bottom up approach).
- Each object has its own internal data and similar objects constitute a class. So each object is a member of a class.
- Class diagram is used to represent the object oriented design. UML modeling and use case are also used in object oriented design.
- Data in the system is not centralized and shared but is distributed among the objects in the system.
- Three main concepts: Encapsulation, Inheritance and Polymorphism greatly enhance the ability of developers' to more easily manage the software product.
- **Encapsulation** combining data and functions into a single entity, **inheritance** provide reusability and by **Polymorphism** same context can be used for different purposes.

- **Advantages of object oriented design:**
  - Reduce maintenance.
  - Provide code reusability, reliability, modeling, reliability and flexibility.
  - Provide robustness to the system.
  - It provides roadmap for reusing objects in new software.
  - The object oriented design process is consistent from analysis, through design to coding.

❖ DATA MODELING CONCEPTS

- A data model is a conceptual relationship of data structure (tables) required for a database. And it is concerned with the structure rather than rules.
- To avoid the redundancy of data in OLTP (online transaction process) database, there is a need to create data model.
- Data model provides abstract and conceptual representation of data.
- Data modeling or ER diagram gives the concepts of objects, attributes and relationship between objects.
- ER diagram is a structured analysis technique. And also describes logical data design that can converted easily into table structure.
- ERD is a snapshot of data structure.
- ER diagram can be used to model the data in the system and how the data items relate to each other but do not cover how the data is to be processed.
- ER diagram enables a software engineer to identify data objects and their relationships using graphical notations.
- ERD is a detailed logical representation of any system. It has three main elements $\rightarrow$ data object (entity), attributes and their relationships.

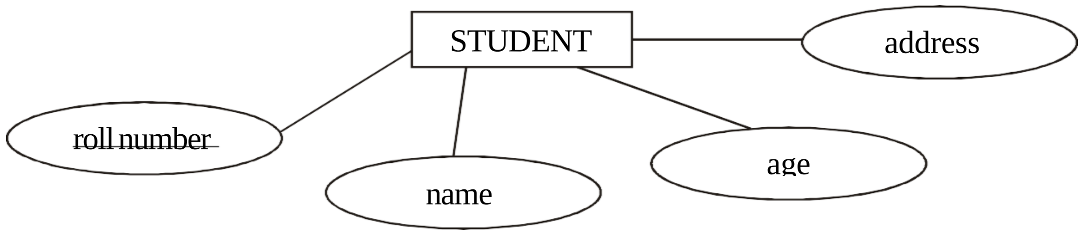
- **Data object (Entity set):**
  - A data object is a real world entity or thing.
  - Data object is a fundamental composite information system.
  - An entity $\rightarrow$ represents a thing that has meaning and about which you want to store or record data.
  - It can be external entity, a thing, an organization, a place or an event.
  - For example: for a college $\rightarrow$ department, students, head of the department may be entities.
  - It has number of properties or attributes. Each object has its own attributes.
  - Entities are represented using rectangle box and preferably entity name is written in capital letters.



Entity set for a college

- **Data Attributes :**
  - An attribute is a property or characteristic of an entity.
  - Attributes provide meaning to the objects.
  - Attributes must be defined as an identifier, and that become key to find instance of object.

- For example: attributes for entity student is roll number, name, age, address etc.
- Attributes represented using oval.

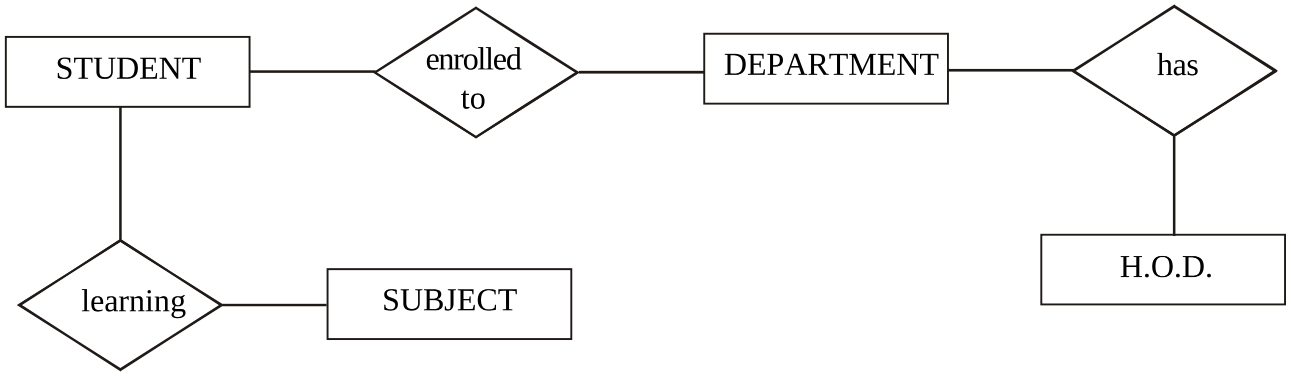


Attributes of entity student

(Note: attribute 'roll number' is underlined because it is key attribute)

➢ Relationship:

- Entities are connected to each other via relations. Generally relationship is binary because there are two entities are related to each other.
- Relationship illustrates how two entities share information in the database structure.
- Relationship of objects is bidirectional, so they can be read in either side.
- Relationship is represented using diamond shape symbol with joined relationship name.
- Below figure gives the understanding of relationship between student, department, head of the department and subject.



Relationship of entities

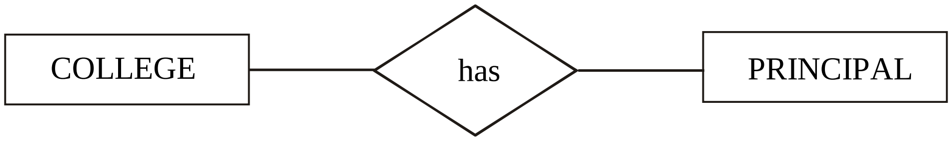
➢ Cardinality :

- The elements of data modeling - data objects, attributes, and relationships, provide the basis for understanding the information domain of a problem. However, additional information related to these basic elements must also be understood.
- Object X has relationship to object Y does not provide enough information for software engineering purposes. We must understand how many occurrences of object X are related to how many occurrences of object Y. This leads to a data modeling concept called cardinality.
- The concept of cardinality defines the maximum number of objects that can participate in a relationship. That means number of occurrences of one [object] that can be related to the number of occurrences of another [object].
- Cardinality is usually expressed as simply 'one' or 'many.' For example a father can have one or more children but a child can have only one father.

- Maximum cardinality means maximum number of instance attached in relationship and minimum in vise versa.
- Different cardinalities are explained below.

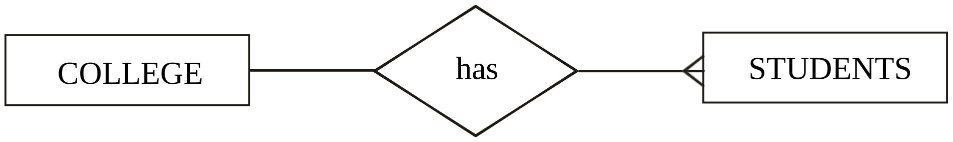
➢ **One to One (1 : 1)**

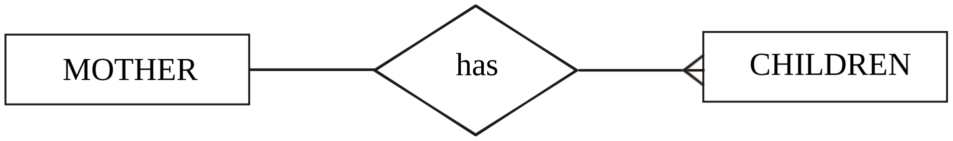
- An instance (occurrence) of entity (object) A can relate to one only instance (occurrence) of B and instance of B can relate to only one instance of A.
- For example one college has only one principal and one principal can take charge of one college only.



➢ **One to Many (1 : M)**

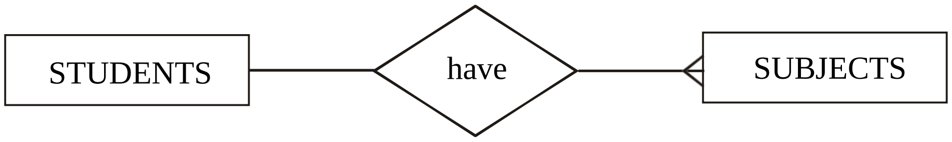
- One instance of entity A can relate to one or many instances of B, but an instance of B can relate to only one instance of entity A.
- A mother can have many children but a child can have only one mother. In another example, one college can have many students but a student can be enrolled to one college.

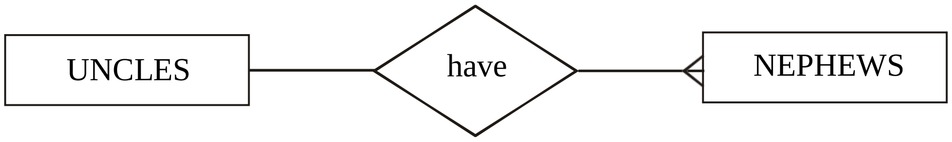




➢ **Many to Many (M : M)**

- One or more instances of entity A can relate to one more instances of entity B. and vice versa.
- For example an uncle can have many nephews while a nephew can have many uncles. In another example, many students can registered with one or more subjects as well one subject can be registered by one or more students.

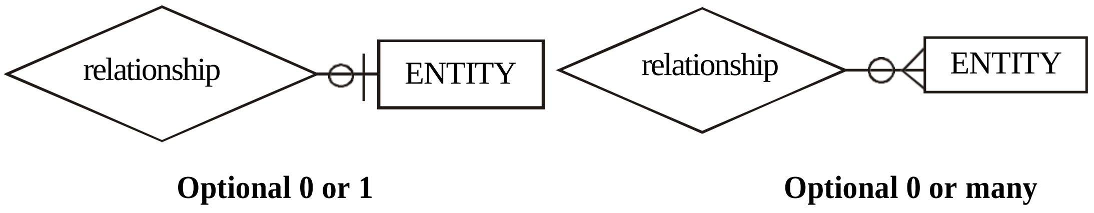




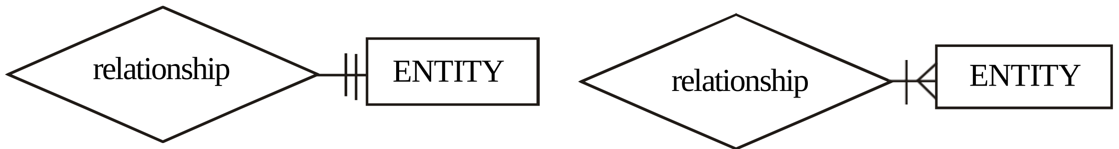
➢ **Modality:**

- Cardinality does not, however, provide an indication of whether or not a particular data object must participate in the relationship. To specify this information, the data model adds modality concept to the object/relationship pair.
- Modality is the form of cardinality.

- Modality means a classification of relationships on the basis of whether they claim necessity, possibility or impossibility.
- The modality of relationship is 0 if there is no explicit need for the relationship or the relationship is 0 or it is optional.
- The modality is 1 if an occurrence of relationship is mandatory.
- The notations for modality are explained below.



Optional 0 or 1 **Optional 0 or many**



Mandatory 1 **Mandatory many**

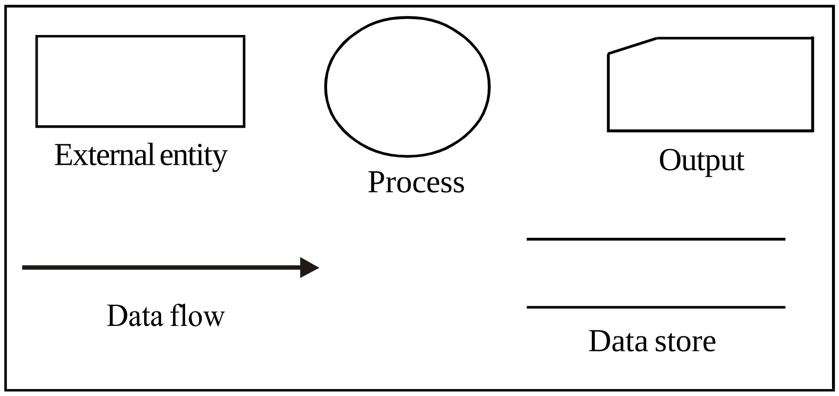
(**Note:** There are many different notations available for cardinality and modality, like Chen's notations, Crow's foot notation etc. So it may be possible that you may see different notations in various books or websites.)

❖ DATA FLOW DIAGRAMS

- DFD (Data Flow Diagram) is also known as bubble chart or data flow graph.
- DFDs are very useful in understanding the system and can be effectively used during analysis. It shows flow of data through a system visually.
- The DFD is a hierarchical graphical model of a system that shows the different processing activities or functions that the system performs and the data interchange among these functions.
- It views a system as a function that transforms the inputs into desired outputs.
- Each function is considered as a process that consumes some input data and produces some output data.
- The system is represented in terms of the input data to the system, various processing carried out on these data, and the output data generated by the system.
- Functional model can be represented using DFD.

➢ Primitive symbols used in construction of DFD model:

- The DFD model uses a very limited number of primitive symbols.



DFD symbols

1. **Process (function)**
   - Process or function is represented by circle or bubble.
   - Circles are annotated with names of the corresponding functions.
   - A process shows the part of the system that transforms inputs into outputs.
     - The process is named using a single word that describes what the system does functionally. Generally processis named using 'verb'.
2. **External entity**
   - Entity is represented by a rectangle. Entities are external to the system which interacts by inputting data to the system or by consuming data produced by the system.
   - It can also define source (originator) or destination (terminator) of the system.
3. **Data flow**
   - Data flow is represented by an arc or by an arrow.
   - It used to describe the movement of the data.
   - It represents the data flow occurring between two processes, or between an external entity and a process. It passes data from one part of the system to another part.
   - Data flow arrows usually annotated with the corresponding data names. Generally data flow named using 'noun'.
4. **Data store**
   - Data store is represented by two parallel lines.
   - It is generally a logical file or database.
   - It can be either a data structure or a physical file on the disk.
5. **Output**
   - Output is used when a hardcopy is produced.
   - It is graphically represented by a rectangle cut either a side.

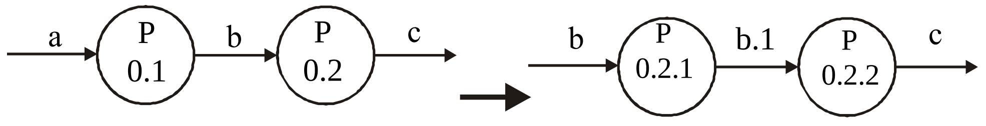
➢ **Developing DFD model of the system:**

- DFD graphically shows the transformation of the data input to the system to the final result through a hierarchy of levels.
- DFD starts with the most abstract level of the system (lowest level) and at each higher level, more details are introduced.
- To develop higher level DFDs, processes are decomposed into their sub functions.
- The abstract representation of the problem is also called context diagram.

➢ **Context diagram (Level 0 DFD)**

- The context diagram is top level diagram; it is the most abstract data flow representation of a system.
- It is an overall, simplified view of target system.
- It only contains one process node that generalizes the function of entire system with external entities. (Itrepresents the entire system as a single bubble.)
- Data input and output are represented using incoming and outgoing arrows. These arrows should be annotated with the corresponding data items (usually a noun).

- **Level 1 diagram**
  - To develop the level 1 DFD, we have to examine the high-level functional requirements.
  - It is recommended that 3 to 7 functional requirements can be directly represented as bubbles in 1st level.
- **Further Decomposition (Level 2 and above)**
  - The bubbles are decomposed into sub-functions at the successive levels of the DFD.
  - Decomposition of a bubble is also known as factoring or exploding a bubble.
  - Each bubble at any level of DFD is usually decomposed between 3 to 7 bubbles in its higher level.
  - Successive levels give more detailed description about the system.
  - It's not a rule that particular number of levels are needed for the system, but decomposition of a bubble should be carried on until a level is reached at which the function of the bubble can be described using a simple algorithm.
- **Numbering the bubbles**
  - It is necessary to number the different bubbles occurring in the DFD.
  - These numbers help in uniquely identifying any bubble in the DFD by its bubble number.
  - The bubble at the context level is usually assigned the number 0 to indicate that it is the 0 level DFD.
  - Bubbles at level 1 are numbered, 0.1, 0.2, 0.3, etc.
- **Balancing DFD**
  - The data that flow into or out of a bubble must match the data flow at the next level of DFD. This is known as balancing a DFD.



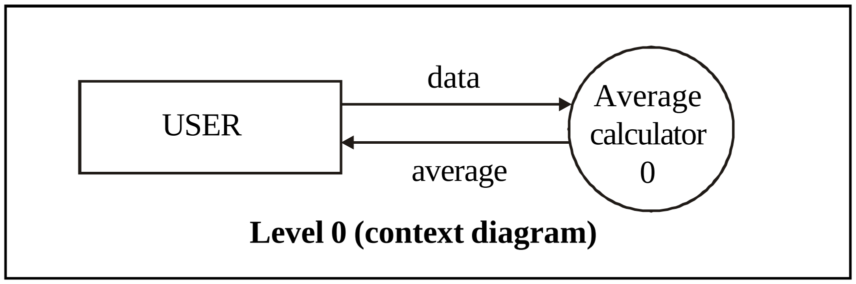
Data flow in level - 1 **Data flow in level - 2**

- **Some care should be taken while constructing DFDs (Guidelines when drawing DFDs) :**

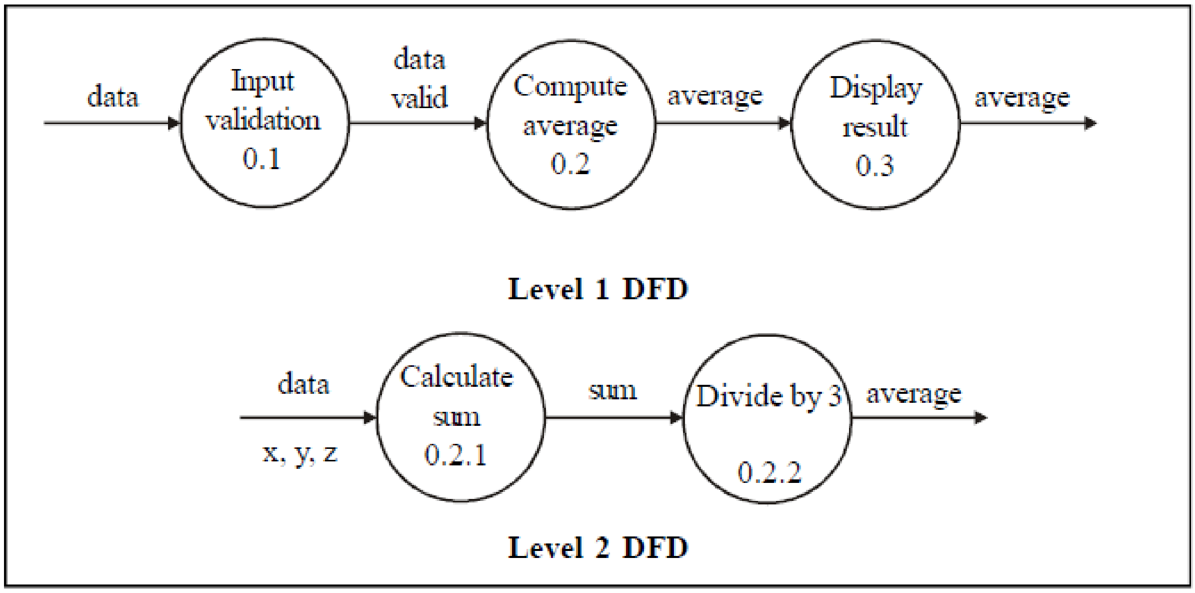
- A process must have at least one input and one output data flow.
- No control information (IF-THEN-ELSE) should be provided in DFD.
- A data store must always be connected with a process. A data store cannot be connected to another data store or to an external entity.
- Data flows must be named.
- Data flows from entities must flow into processes, and data flows to entities must come from processes.
- There should not be detailed description of process in context diagram.
- Name of the data flow should be noun and name of process should be verb.
- Each low level DFD must be balanced to its higher level DFD (input and output of the process must be matched in next level).
- Data that travel together should be one data flow.
- No need to draw more than one bubble in context diagram.
- Generally all external entities interacting with the system should be represented only in the context diagram.

- Be careful with number of bubbles in particular level DFD, as too less or too many bubbles in DFD are oversight.
- All the functionalities of the system specified in SRS must be captured by the DFD model.

➤ **Simple DFD example of average calculator of three numbers.**



Level 0 (context diagram)

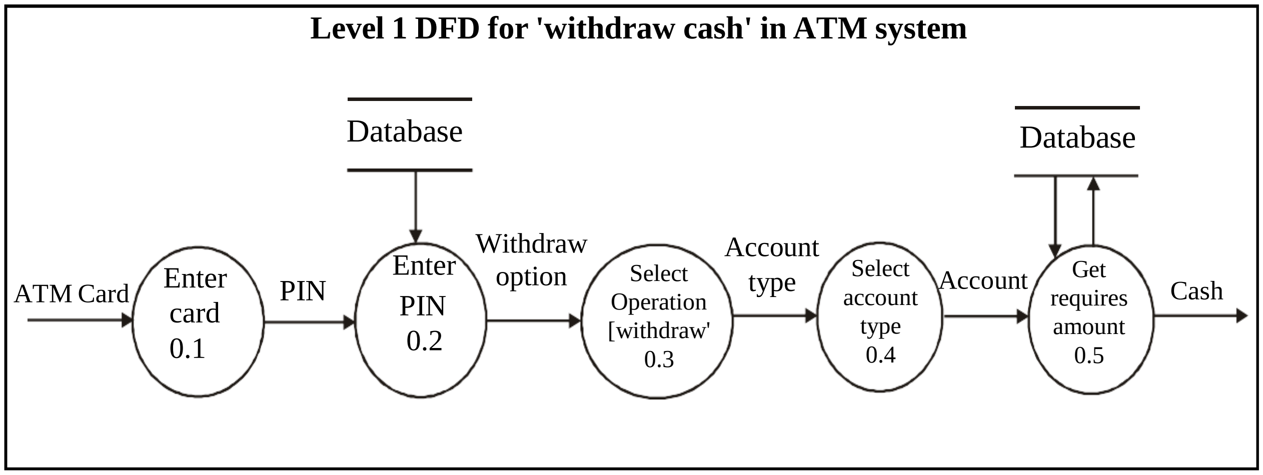


Level 1 DFD

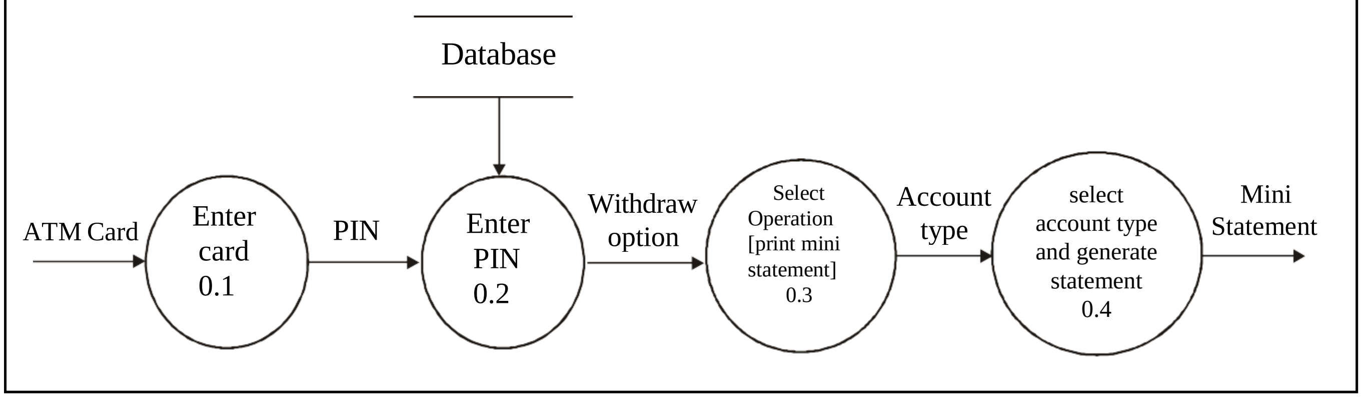
Level 2 DFD

➤ **Another example of ATM system.**

Level 1 DFD for 'withdraw cash' in ATM system



Level 2 DFD for 'print mini statement' in ATM system


Database
ATM Card
Enter card 0.1
PIN
Enter PIN 0.2
Withdraw option
Select Operation [print mini statement] 0.3
Account type
select account type and generate statement 0.4
Mini Statement

Note: Level1 DFD can be drawn for each high level functional requirement shown in the SRS.

Advantages of DFD model:

- DFD is a simple graphical technique which is very simple to understand and easy to use.
- It can be used as a part of system documentation.
- DFD can provide detailed description of the system components.
- It provides clear understanding to the developers about the system boundaries and analysis of the system.
- It explains the logic behind the data flow within system.
- It provides structure analysis of the system.
- Symbols used in DFD model are very less.
- It does not provide any time dependent behavior like we cannot consider at which time we have to do particular process.

Disadvantages of DFD model:

- Control information is not defined by a DFD.
- DFD does not provide any specific guidance as how exactly decompose a given function into its sub functions; we have to use subjective judgment to carry out decomposition.
- Sometimes it puts programmers in little confusing state.
- Different DFD models have different symbols, e.g. in Gane and Sarson notations → process is represented as rectangle while in Demarco and Yordan notations → it is ellipse, so making confusion at the time of referring. (In some of the notations you can see the process symbol is rectangle while in some other notations, process symbol is ellipse and that is confusing).
- Physical considerations are left out in DFD.

❖ SCENARIO BASED MODELING

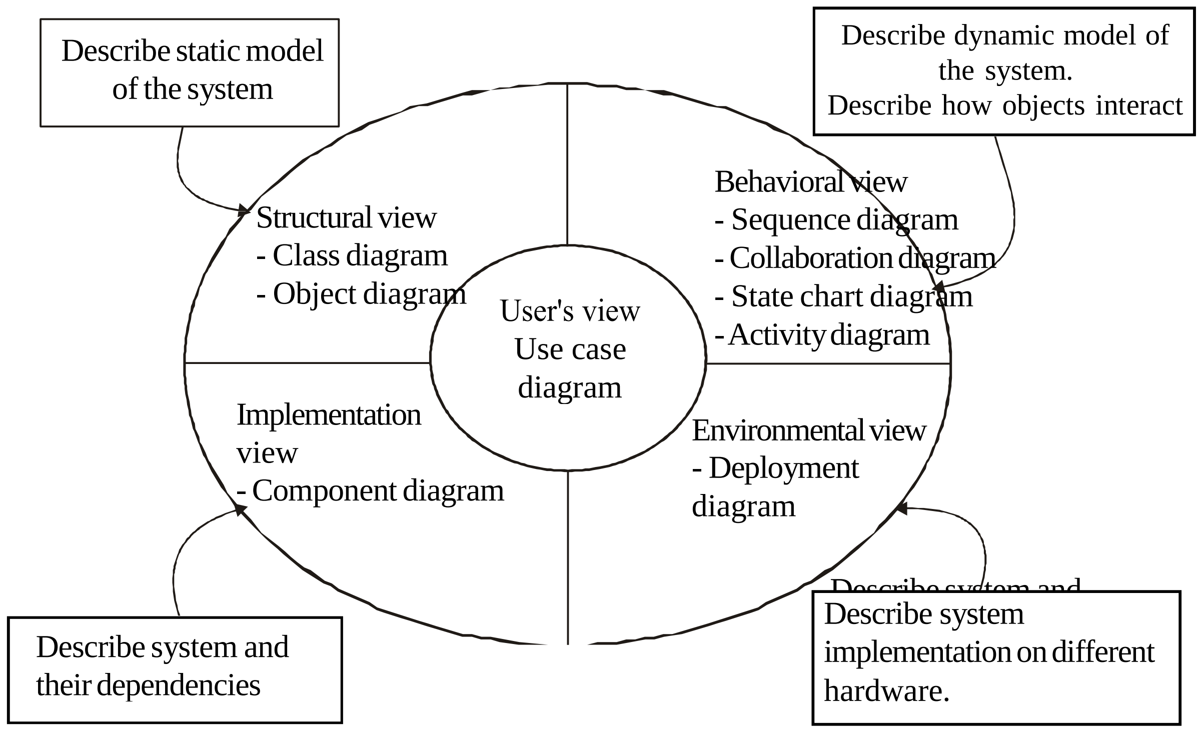
- A scenario describes a set of actions that are performed to achieve some specific conditions. And this set is specified as a sequence.
- Each step in scenario is a logically complete action performed either by the actor or by the system.

➢ UML (Unified Modeling Language):

- As the name suggests, UML is a modeling language.
- UML is a general purpose modeling language in the field of software engineering, which is designed a standard way to visualize the design of the system.
- The goal is for UML to become a common language for creating models of object oriented computer software.
- UML is very useful in documenting the design and analysis results.
- It may be used to visualize, specify, construct, and document the software system.
- UML is not a design methodology.
- UML making system easy to understand using less number of primitive symbols.

➢ UML diagrams

- UML can be used to construct nine different types of diagrams to capture five different views of a system.
- UML diagrams provide different perspectives of the software system.


Describe static model of the system
Describe dynamic model of the system. Describe how objects interact
Structural view - Class diagram - Object diagram
Behavioral view - Sequence diagram - Collaboration diagram - State chart diagram - Activity diagram
User's view Use case diagram
Implementation view - Component diagram
Environmental view - Deployment diagram
Describe system and their dependencies
Describe system implementation on different hardware.

UML diagrams

➢ Writing Use-Cases (Use-Case diagram):

- Use case model in UML provides system behavior.
- The use case model for any system consists of a set of "use cases".
- Use cases represent the different ways in which a system can be used by the users.

- **Use case diagram is a representation of a user's interaction with the system that shows the relationship between the users and different use cases in which user is involved.**

- The purpose of a use case is to define the logical behavior of the system without knowing the internal structure of it.
- Use case identifies the functional requirements of the system.
- Use case diagram describes "who can do what in a system".
- A use case typically represents a sequence of interactions between the user and the system.
- Use cases corresponding to the high level functional requirements.
- For example $\rightarrow$ in ATM system, the system should have following use cases.
  - ➢ Check balance
  - ➢ Withdraw money
  - ➢ Deposit money
  - ➢ Transfer money
  - ➢ Print mini statement
  - ➢ Link aadhar card
  - ➢ Pin change etc.

➢ Components of use case diagram (Representation of use case diagram)

- Three main components along with relationships are used in use case diagram.

I. Use case

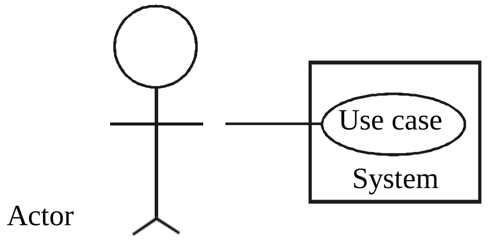
- Each use case is represented by an ellipse with the name of the use case written inside the ellipse, named by verb.
- All the use cases are enclosed with a rectangle representing system boundary. Rectangle contains the name of the system.
- It identifies, clarifies and analyzes the functional requirements of the system.

II. Actor

- An actor is anything outside the system that interacts with it, named by noun.
- Actors in the use case diagram are represented by using the stick person icon.
- An actor may be a person, machine or any external system.
- In use case diagram, actors are connected to use cases by drawing a simple line connected to it. Actor triggers use cases.

- **Each actor must be linked to at least one use case, while some use cases may not be linked to actors.**

- A simple figure showing the use cases and actor in the system.



III. Relationship

- Relationships are represented using 'a line' between an actor and a use case. It is also called communication relationship.
- An actor may have relationship with more than one use case and one use case may relate to more than one actor.

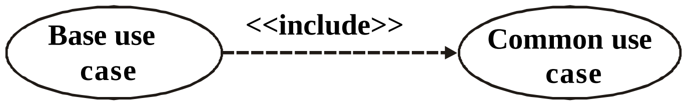
➢ **Different relationships in use case diagram are explained below:**

1. Association

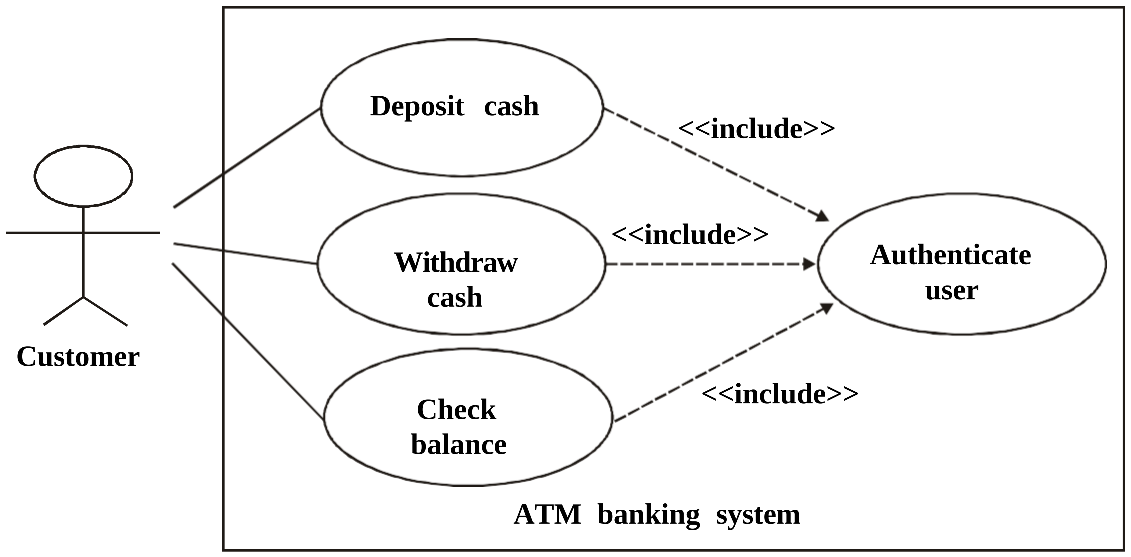
- This relationship is the interface between an actor and a use case.
- It is represented by joining a line from actor to use case.

2. Include relationship

- It involves one use case including the behaviour of another use case.
- The "include" relationship occurs when a chunk of behaviour that is similar across a number of use cases.
- It is represented using predefined stereotype <<include>>.
- An include relationship is depicted with a directed arrow having a dotted line. The tip of arrowhead points to the child use case and the parent use case connected at the base of the arrow.



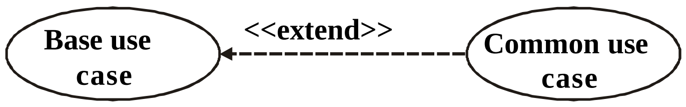
- Example of include relationship is showing in below figure.



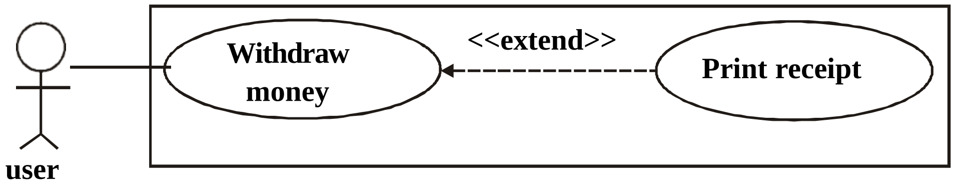
3. Extend relationship

- It allows you to show optional behavior of the system. Extend is optional and supplementary.
- This relationship among use cases is represented as a stereotype <<extend>>.
    - It is depicted with a directed arrow having a dotted line. The tip of arrowhead points to the base use case andthe child use case is connected at the base of the arrow.
    - Extend relationship exists when one use case calls another use case under certain condition (like: If - thencondition).

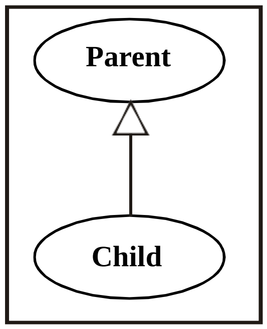
- Extend relationship is optional and supplementary.



- Example of include relationship is showing in below figure.

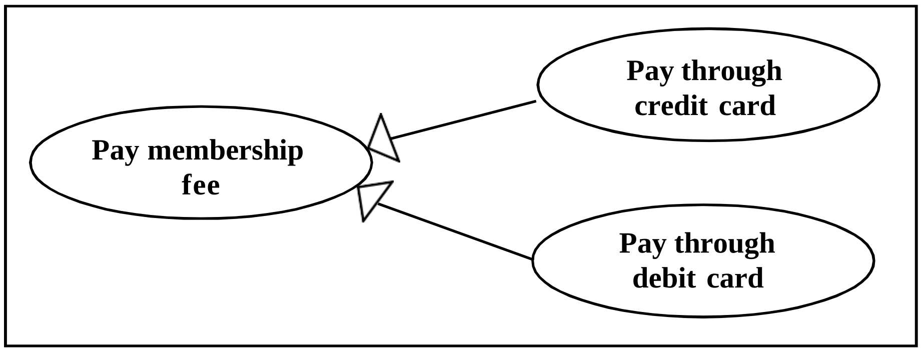


4. **Generalization**



- A generalization relationship is a parent-child relationship between use cases.
- Use case generalization can be used when you have one use case that is similar to another, but does slightly different.
- The child use case is an enhancement of the parent use case.
- Generalization is shown as a directed arrow with a triangle arrowhead. The child use case is connected at the base of the arrow. The tip of the arrow is connected to the parent use case.
- The child may override the behavior of its parent.

Example of generalization is showing in below figure.

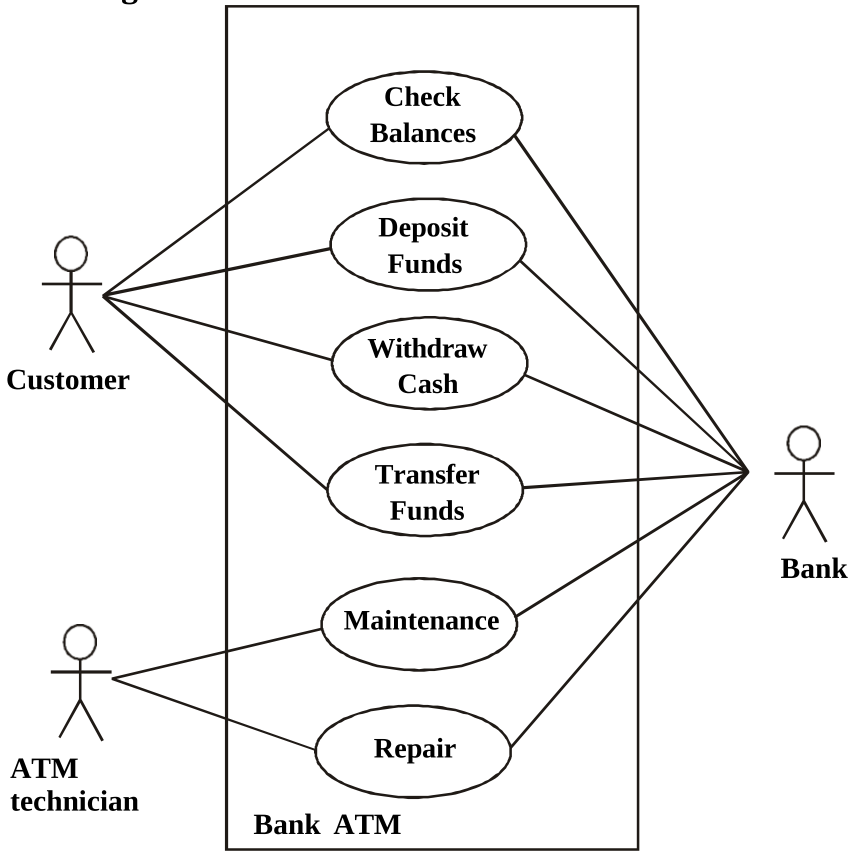


Use case guidelines

- Identify all different users. Give suitable names.
- For each user, identify tasks. These tasks will be the use cases.

- Use case name should be user perspective.
- Show relationships and dependencies.

➢ **Example: Use case diagram for bank ATM**



➢ **Advantages of use case diagram**

- It is easy to understand and draw.
- It is used to capture the functional requirements of the system.
- It is used as basis for scheduling effort.
- It is used to verify whether all the requirements are captured.

➢ **Disadvantages of use case diagram**

- Use case diagrams are not fully object oriented.
- It does not provide any guideline when to stop.

➢ **Applications of use case diagram**

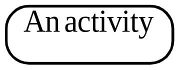
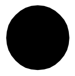
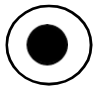
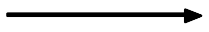
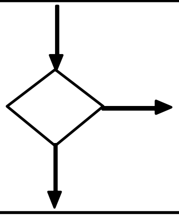
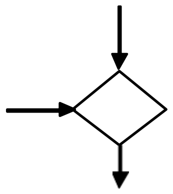
- Requirement analysis
- High level design
- Reverse engineering
- Forward engineering

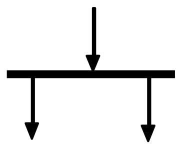
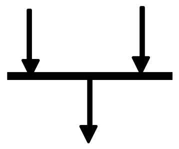
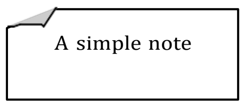
➢ **Developing an Activity diagram:**

- An activity diagram falls under the category of behavioral diagram in UML.
- An activity diagram is a UML diagram that is used to model a process. It models the actions (behavior) performed by the system components, the order in which the actions take place and conditions related to actions.

- Activity Diagrams consist of activities, states and transitions between activities and states.
- It describes how the events in a single use case relate to one another.
- The aim of activity diagram is to record the flow of control from one activity to another of each actor and to show interaction between them.
- It mainly represents series of actions and flow of control of a system.
- It focuses on the how of activities involved in a single process. Also shows how activities depend on one another.
- Activity diagrams represent workflows in a graphical way.
- Activity diagrams are similar to procedural flow charts. The difference is that activity diagram support parallel activities.
- An activity is a state with an internal action and one or more outgoing transitions.
- An interesting feature of the activity diagrams is the swimlanes. It enables you to group activities based on who is performing them. So, swimlanes make group of activities based on actors.

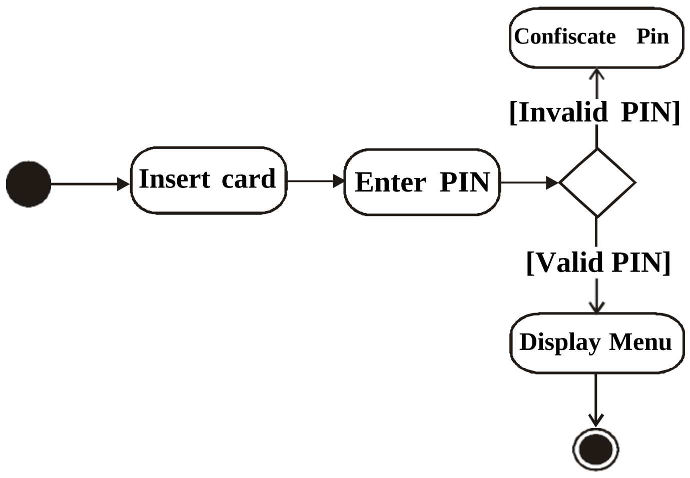
➢ Elements (components) of an activity diagram

Elements or Components and its description	Symbol
Activity – It represents a particular action taken in the flow of control. – It is denoted by a rectangle with rounded edges. And labelled inside it describing corresponding activity. – There are two special type of activity nodes : 1. **Initial activity (OR Start activity)** – This shows the starting point or first activity of the flow. – It is denoted by a solid circle. 2. **Final activity (OR End activity)** – The end of the activity diagram shown by a bull's eye symbol. It represents the end point of all activities.	  
Flow or Transition – A flow (also termed as edge or transition) is represented with a directed arrow. – This is used to show transfer of control from one activity to another.	
Decision (Branch) – A decision node represented with a diamond. – It is a branch where single transition (flow) enters and several outgoing transitions.	
Merge – This is represented with a diamond shape with two or more input transitions and a single output transition.	

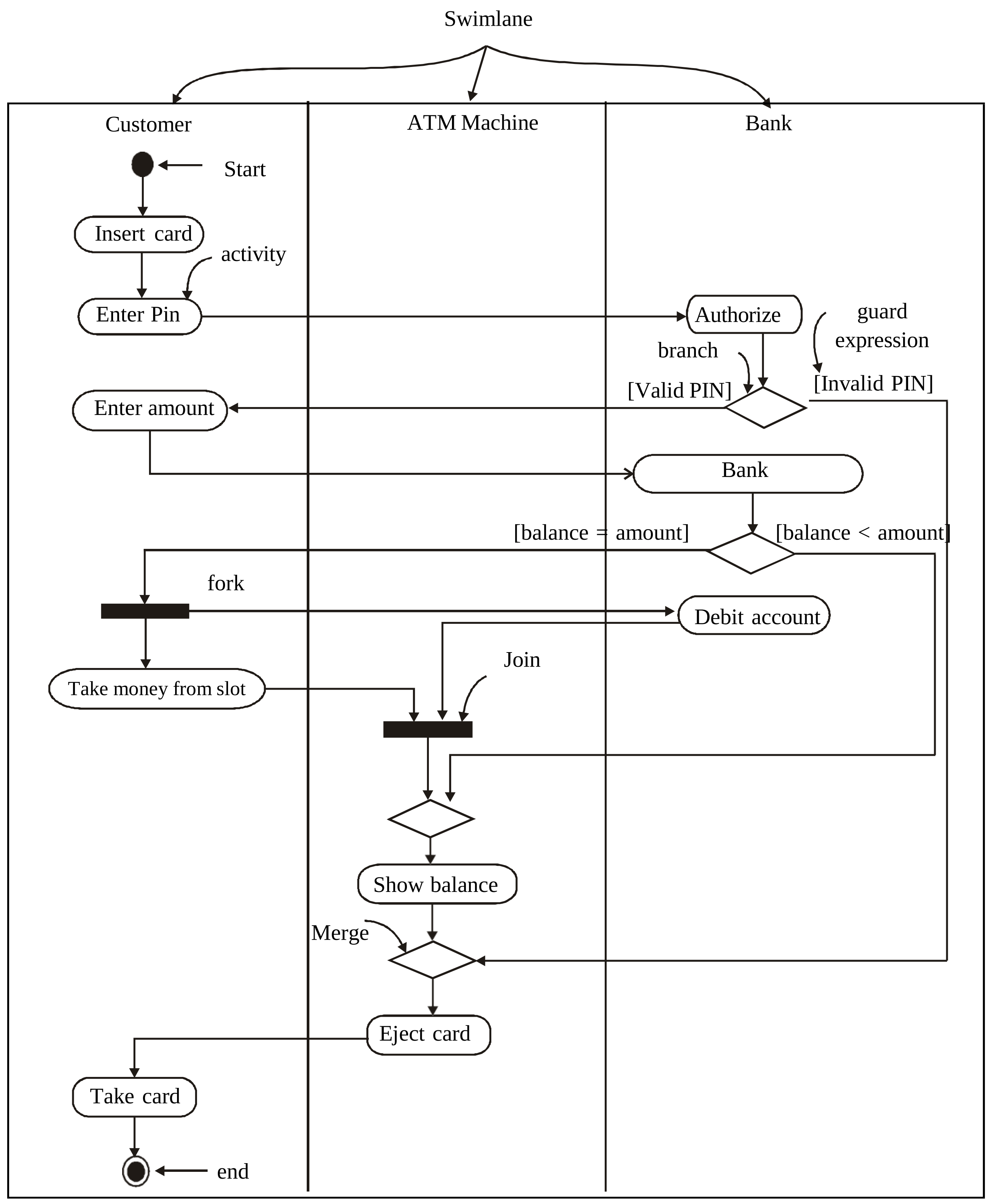
Fork – Fork is a point where parallel activities begin. – Fork is denoted by black bar with one incoming transition and several outgoing transitions. – When the incoming transition is triggered, all the outgoing transitions are taken Into parallel.	
Join – Join is denoted by a black bar with multiple incoming transitions and single outgoing transition. – It represents the synchronization of all concurrent activities.	
Note – UML allows attaching a note to different components of diagram to present some textual information. – It could be some comments or may be some constraints. – A note generally attached to a decision point to indicate the branching criteria. – It is denoted by a rectangle with cut a side.	
Partition or Swimlanes – Different components of an activity diagram can be logically grouped into different areas, called partition or swimlanes. – They often correspond to different users or different units of organization. – It is denoted by drawing vertical parallel lines. – Partitions in an activity diagram are not mandatory.	
Guard conditions – Guard conditions control transition from alternative transitions based on condition. – These are represented by square brackets.	**[ ]**

Typical symbols used in activity diagram

➢ **A simple example activity diagram (ATM system)**



- **Another example showing swimlanes**



- **Advantages of activity diagram**
  - Activity diagrams can be very useful to understand complex processing activities.
  - Different activities are grouped together based on actor. That is represented by swimlanes.
  - It can be useful for analyzing a use case and understating workflow of system.

- Activity Diagrams are good for describing synchronization and concurrency between activities.
- Partitioning can be helpful in investigating responsibilities for interactions and associations between objects andactors.

➢ **Disadvantages of activity diagram**

- The activity diagram does not provide message part. Means do not show any message flow from one activityto another.
- It can't describe how objects collaborate.
- It takes time to implement.
- Complex conditional logics (like Truth table) can't be represented by activity diagram.
- There are lot many symbols compare to other UML diagrams, so sometimes make it confusing to the developer.

➢ **When to use Activity diagram (Applications of activity diagram)**

- Mainly activity diagram used to describe the parallel behavior of the system. It makes a great tool for workflowmodeling.
- It is also used in multithreaded programming application.

➢ **Difference between flowchart and activity diagram**

Flow chart	**Activity diagram**
It is limited for sequential access.	It is used for parallel and concurrent processing.
It is used for flow of control through an algorithm, not used for object oriented procedure.	It is usually used for object oriented systems.
Concept of swimlanes is not there in it.	It has the functionality of swimlanes.
It has limited functionalities compare to activity diagram.	It has more functionality.