Unit-II
Software Requirement Analysis and Design

Prepared By : H.C.Savaliya
Computer Department
AVPTI Rajkot

Course Outcomes

Prepare software analysis and design using SRS, DFD and object oriented UML diagrams.

Unit Outcomes

- Identify Software requirements for the given problem
- Prepare SRS from the requirement analysis
- Represent the specified problem in the given design notation – DFD
- Draw the relevant UML diagrams for the given problem

Requirements analysis and specification

- The requirements analysis and specification phase starts once the feasibility study phase is complete and the project is found to be financially and technically feasible.
- Goal of this phase is to clearly understand the customer requirements and to systematically organize these requirements in a specification document.
- This phase consist of the two activities:
  - Requirements gathering and analysis
  - Requirements specification

Requirements gathering and analysis

- **Requirements Gathering :** it involves interviewing the end users and customers and studying the existing documents to collect all possible information of the system.
- If there is no old working system is present than this task is required more imagination and creativity.

- **Analysis of gathered requirements :** the main purpose of this activity is to clearly understand the exact requirements of the customer
- The following basic questions can be used to understand exact requirement.

Continue…

- What is the problem?
- Why is it important to solve the problem?
- What are the possible solutions to the problem?
- What exactly are the data input to the system and what exactly are the data output by the system?
- What are the likely complexities that might arise while solving the problem?
- If there are external software or hardware with which the developed software has to interface, then what exactly would the data interchange formats with the external system be?

Continue...

- After the analyst has understood the exact customer requirements, he proceeds to identify and resolve the various requirements problems.
- The most important requirements problems that the analyst has to identify and eliminate are the problems of **inconsistencies and incompleteness**.
- When the analyst detects any inconsistencies or incompleteness in the gathered

  requirements, he resolves them by carrying out further discussions with the end users

and the customers.

Software Requirements Specification (SRS)

- After the software analyst has collect all information and remove all problems than he starts to systematically organize the requirements in the form of SRS document.
- SRS Document usually contains all the user Requirements. it is a contract document between customer and development organization.
- Once the customer agree to SRS Document the development team starts to develop the product and implement all functionalities which are specified in SRS document.
- There are many types of users of SRS Document during software development process.

Important Categories of users of SRS Document

- Users ,Customers and marketing personnel.
- Software developers.
- Test Engineers.
- User Documentation writers.
- Project managers.
- Maintaince engineers

Contents of the SRS Document

- The important parts of SRS document are:
- ❑ Functional requirements of the system
- ❑ Non-functional requirements of the system
- ❑ Goals of implementation

- **Functional requirements:-**
- The functional requirements part discusses the functionalities required from the system.
- The system is considered to perform a set of high-level functions {*fi*}.

Continue…

- Each function fi of the system can be considered as a transformation of a set of input data (i) to the corresponding set of output data (o). The user can get some meaningful piece of work done using a high-level function

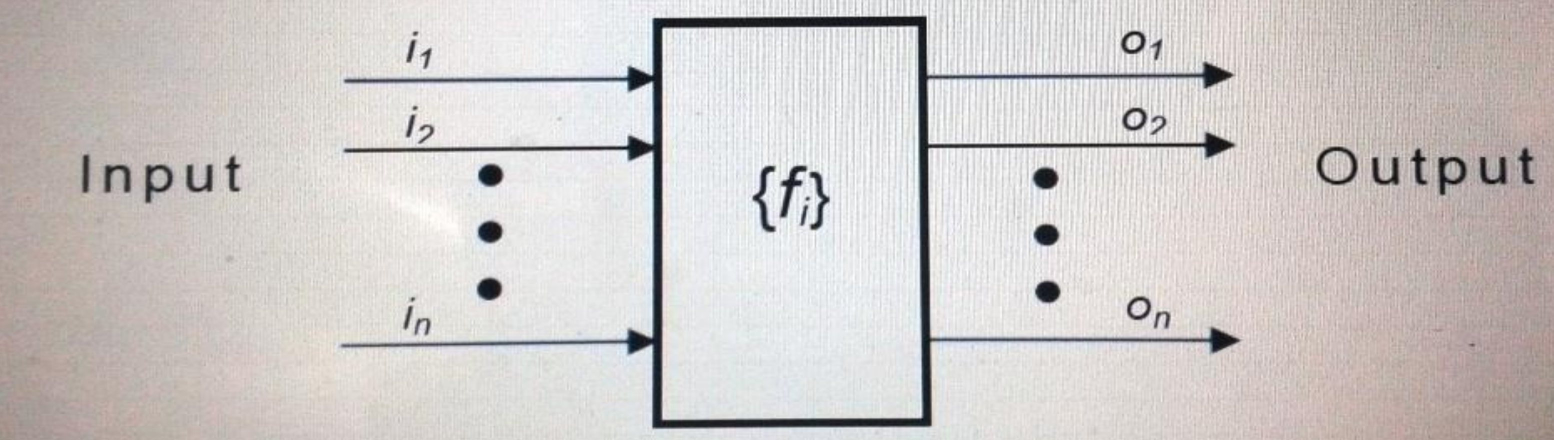


Fig. 3.1: View of a system performing a set of functions

Nonfunctional requirements

- Nonfunctional requirements deal with the characteristics of the system which can not be expressed as functions – such as the maintainability of the system, portability of the system, usability of the system, etc.

- **Nonfunctional requirements may include:**

- Reliability
- Accuracy
- Human – Computer interface,
- Security,Portability,Availability,Usability etc.

Goals of implementation

- The goals of implementation part documents some general suggestions regarding development.
- The goals of implementation section might document issues such as revisions to the system functionalities that may be required in the future, new devices to be supported in the future, reusability issues, etc.
- These are the items which the developers might keep in their mind during development so that the developed system may meet some aspects that are not required immediately.

Continue...

- So if the requirements in form of the function (input and output) than you can documented as a functional requirements.

- Any other requirements which can be verified by inspecting the system, are documented as a non-functional requirements.

- The suggestions for the developers , are documented as goals of the implementation.

Identifying functional requirements from a problem description

- The high-level functional requirements often need to be identified either from problem description or from a conceptual understanding of the problem.
- Each high-level requirement means to perform some meaningful piece of work.
- There can be many types of users of a system and their requirements from the system may be very different. So, it is often useful to identify the different types of users who might use the system and then try to identify the requirements from each user.

Continue...

- Here we list all functions {fi} that the system performs. Each function fi as shown in fig.is considered as a transformation of a set of input data to some corresponding output data.

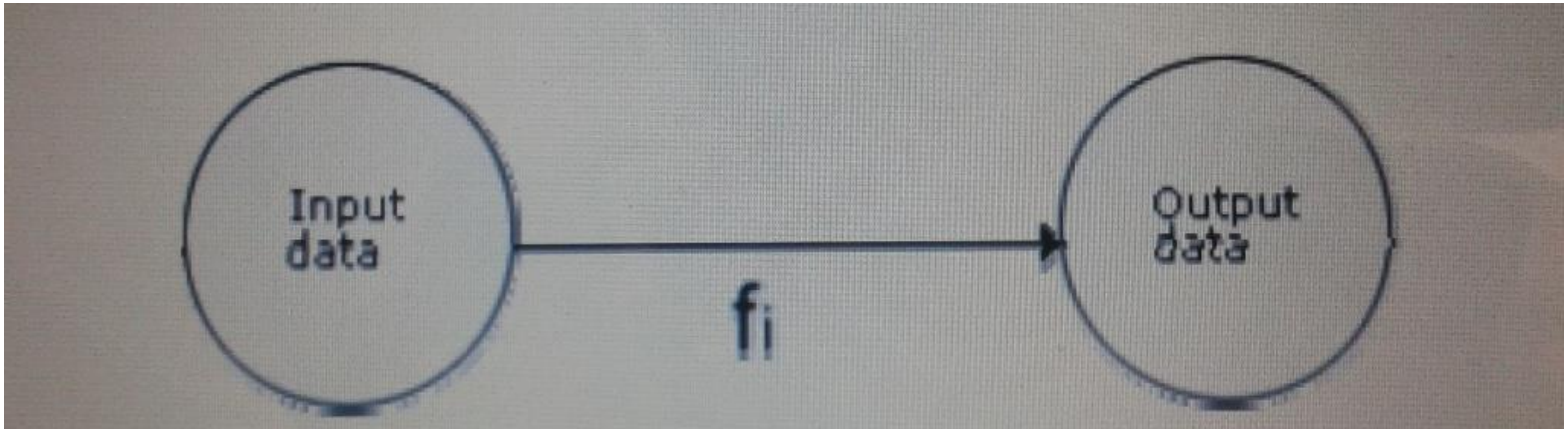


Fig. 3.2: Function f_i

Continue…

- **Example:** Consider the case of the library system, where
- F1: Search Book function
- Input: an author’s name
- Output: details of the author’s books and the location of these books in the library

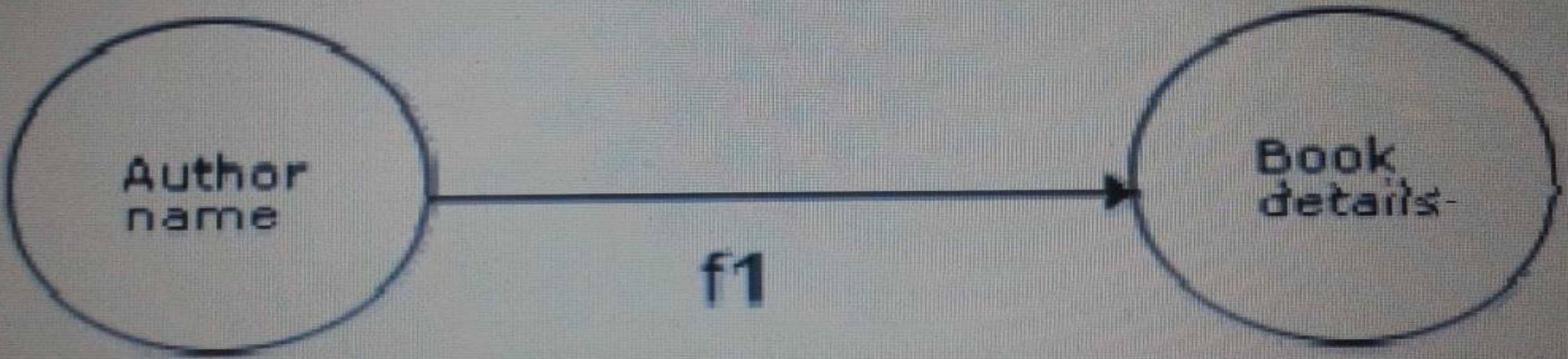


Fig. 3.3: Book Function

Continue...

- So the function Search Book (F1) takes the author's name and transforms it into book details.
- Functional requirements actually describe a set of high-level requirements, where each high-level requirement takes some data from the user and provides some data to the user as an output.
- Also each high-level requirement might consist of someother functions.

Documenting functional requirements

- For documenting the functional requirements, we need to specify the set of functionalities supported by the system.
- A function can be specified by identifying the input to the system(input domain) , and the type of processing to be carried on the input data to obtain the output data, the output data domain.
- Let us first try to document the withdraw-cash function of an ATM (Automated Teller Machine) system.
- The withdraw-cash is a high-level requirement. It has some sub-requirements corresponding to the different user interactions. These different interaction sequences capture the different scenarios.

Continue…

- **Example: – Withdraw Cash from ATM**
- R1: withdraw cash
- Description: The withdraw cash function first determines the type of account that the user has and the account number from which the user wishes to withdraw cash. It checks the balance to determine whether the requested amount is available in the account. If enough balance is available, it outputs the required cash, otherwise it generates an error message.

Continue…

- R1.1 select withdraw amount option

  Input: "withdraw amount" option

  Output: user prompted to enter the account type.
- R1.2: select account type

  Input: user option

  Output: prompt to enter amount.
- R1.3: get required amount

  Input: amount to be withdrawn in integer values greater than 100 and less than10,000 in multiples of 100.

  Output: The requested cash and printed transaction statement.

  Processing: the amount is debited from the user's account if sufficient balance is available, otherwise an error message displayed.

Functional v/s Non functional requirements

Functional Requirements (Verbs)	Non Functional Requirements (Attributes)
Functional requirements help to understand the functions of the system.	They help to understand the system's performance.
Mandatory	Not Mandatory
It concentrates on the user's requirement.	It concentrates on the expectation and experience of the user.
These requirements are specified by the user.	These requirements are specified by the software developers, architects, and technical persons.
Describe what the product does	Describes how the product works
Ex. Login ,Registration, Payment	Ex.Security,Relaibility,Performance etc.

Characteristic of Good SRS Document

The important properties of a good SRS document are the following:

- **Concise** : The SRS document should be concise and at the same time unambiguous, consistent, and complete. irrelevant descriptions reduce readability and also increase error possibilities.
- **Structured** : It should be well-structured. A well-structured document is easy to understand and modify. Therefore, in order to make the modifications to the SRS document easy, it is important to make the document well-structured.

Continue…

- **Black-box view** : It should only specify what the system should do and not specify how to do these. This means that the SRS document should specify the external behavior of the system. The SRS document should view the system to be developed as black box. For this reason, the SRS document is also called the black-box specification of a system.
- **Conceptual integrity** : It should show conceptual integrity so that the reader can easily understand it.
- **Verifiable** : All requirements of the system as documented in the SRS document should be verifiable.

Example of Bad SRS Documents

- Most important problems are incompleteness, ambiguity and contradiction
- Following are the some other category of problems that occur in many SRS documents

- **Over Specification** :over specification means when you try to address ‘how to’ aspects in SRS document.

- **Forward references** :you should not refer to aspects that are discussed much later in the SRS document.

- **Wishful thinking** :The type of problem aspects which would be difficult to implement

Problems without a SRS document

The important problems that an organization would face if it does not develop an SRS document are as follows :

- Without developing the SRS document, the system would not be implemented according to customer needs.
- Software developers would not know whether what they are developing is what exactly required by the customer
- Without SRS document, it will be very much difficult for the maintenance engineers to understand the functionality of the system.
- It will be very much difficult for user document writers to write the users' manuals properly without understanding the SRS document.

Software Design

- Software design deals with transforming the customer requirements, as described in the SRS document, into a form (a set of documents) that is suitable for implementation in a programming language
- The following item must be designed during the design phase :
- Different modules and relationship between modules
- Interface among the modules.
- Data base and Data structure for individual modules
- Algorithm to implement individual modules
- So in design phase SRS document as the input and all the above document is output.

Continue…

- A good software design is not a single step procedure but require several iterations.
- Design activities can be broadly classified into two important parts (**Classification of Design Activities**):
  - Preliminary (or high-level) design and
  - Detailed design.

Preliminary and detailed design activities

- High-level design means identification of different modules and the control relationships among them and the interfaces among these modules.
- The outcome of high-level design is called the program structure or software architecture.
- During detailed design, database ,data structure and the algorithms of the different modules are designed.
- The outcome of the detailed design stage is usually known as the module-specification document.

Design Methodology

- Design methodology provide guidelines for the design activity.
- It depends on the following factors.

Type of software

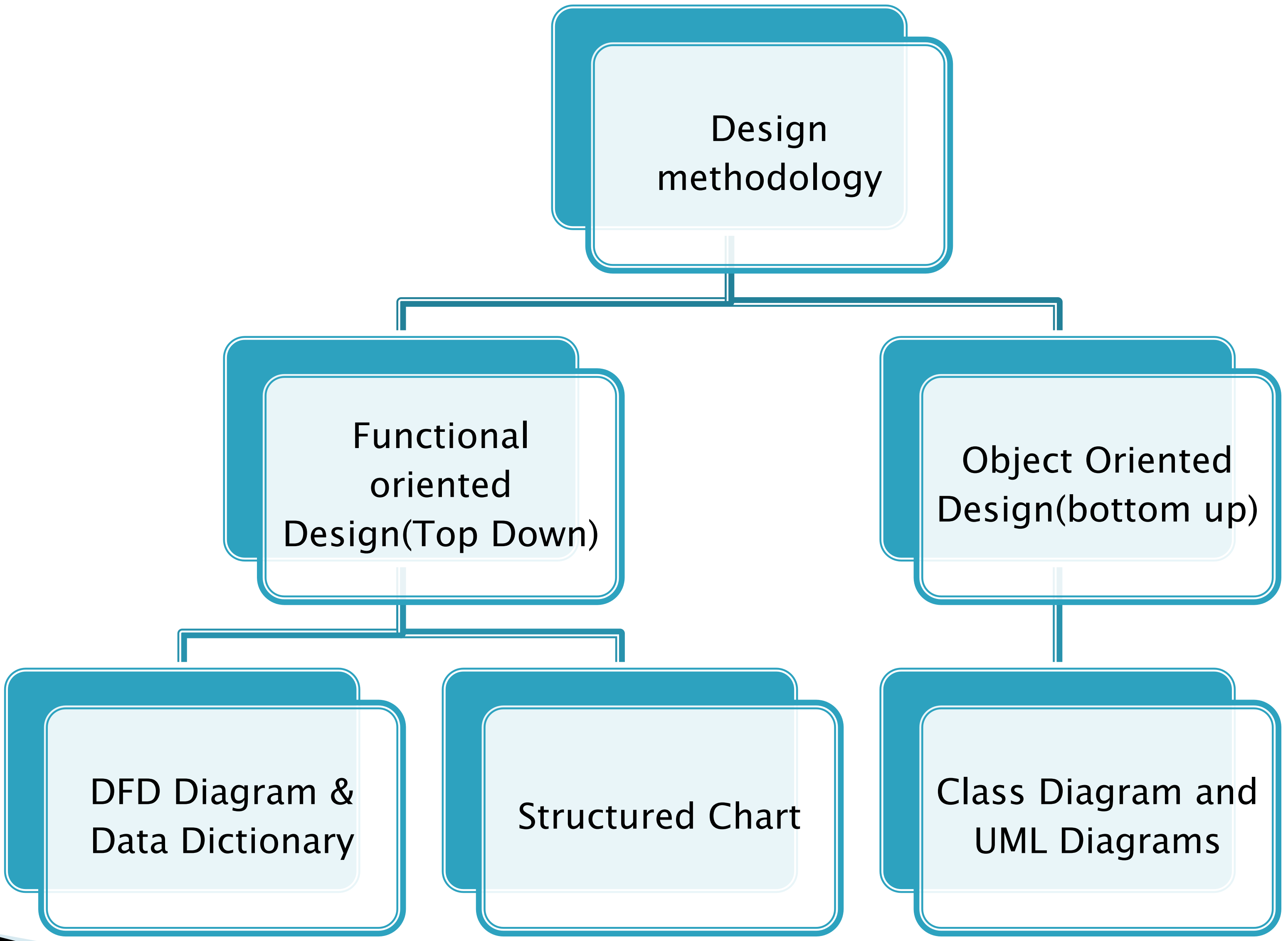
Software development environment

User requirements

Qualification of development team

Available software and hardware resources.

- There are many methodologies for software design for different types of problems.


Design methodology
Functional oriented Design(Top Down)
Object Oriented Design(bottom up)
DFD Diagram & Data Dictionary
Structured Chart
Class Diagram and UML Diagrams

Characteristics of a good software design

- The definition of “a good software design” can vary depending on the application being designed. For example, the memory size used by a program may be an important issue to for a good solution for embedded software development.
- For embedded applications, the other factors of design are not important.
- Therefore, the criteria used to judge the good design depends upon the application.

Continue...

- However, the general features of good design that must be in all types of software.
- **Correctness**: A good design should correctly implement all the functionalities identified in the SRS document.
- **Understandability**: A good design is easily understandable.
- **Efficiency**: It should be efficient.
- **Maintainability**: It should be easily amenable to change.

Continue...

- Possibly the most important goodness criterion is design correctness.
- Second important goodness criterion is understandability of a design.
- A design that is easy to understand is also easy to develop, maintain and change.

Continue…

In order to facilitate understandability, the design should have the following features(**Features of a Design Document**) :

- It should use consistent and meaningful names for various design components.
- The design should be modular. The term modularity means that it should use a cleanly decomposed set of modules.
- It should neatly arrange the modules in a hierarchy, e.g. in a tree-like diagram.
- In short **Modularity ,clean decomposition ,Neat arrangement** are three main features of any software design.

Analysis v/s Design

Analysis	Design
Focuses on determining **what** the business needs are	Design phase takes those business needs and determines **how** they will be met through a specific system implementation
Document the **features** and **functions** the system must have	Figure out **how** to create a system technically that will provide all those needed features and functions.
Emphasizes an **investigation** of the problem and requirements, rather than a solution	Emphasizes a **conceptual solution** that fulfills the requirements, rather than its implementation.
Describe **what** the system supposed to do.	Focus on **how** system should be constructed to satisfy these requirements

Cohesion And Coupling

- Good software design means that clean decomposition of the problem into modules, and the neat arrangement of these modules in a hierarchy.
- The primary characteristics of neat module decomposition are **high cohesion** and **low coupling**.
- Cohesion is a measure of functional strength of a module where as the coupling between two modules is a measure of the degree of interdependence or interaction between the two modules.
- A module having high cohesion and low coupling is said to be functionally independent of other modules. By the term functional independence, we mean that a cohesive module performs a single task or function. A functionally independent module has minimal interaction with other modules.

Continue…

- Functional independence is a key to any good design due to the following reasons :

- **Error isolation** is easy so one error in one module not directly affect other module.

- **Scope for reuse** become possible.

- **Understandability** is easy because all module are independent so complexity is less.

Classification of cohesion

- The different classes of cohesion that a module may contain are shown in the following fig.

Coinci-dental	Logical	Temporal	Proce-dural	Communi-cational	Sequen-tial	Funct-ional

High (Best) $\longrightarrow$ Low (Worst)

Fig. – Classification of Cohesion

Continue…

- **Coincidental cohesion**: A module is said to have coincidental cohesion, if it performs a set of tasks that relate to each other very loosely.
- **Logical cohesion:** A module is said to be logically cohesive, if all elements of the module perform similar operations.
- **Temporal cohesion:** when all functions of module execute in same time span than we can say module contain temporal cohesion
- **Procedural cohesion:** when set of functions of the module are all part of a same procedure (algorithm) than we can say module contain **Procedural** cohesion

Continue…

- **Communicational cohesion:** A module is said to have communicational cohesion, if all functions of the module refer to or update the same data structure, e.g. the set of functions defined on an array or a stack.
- **Sequential cohesion:** if the elements of a module form the parts of sequence, where the output from one element of the sequence is input to the next than we can say module contain **Sequential** cohesion
- **Functional cohesion:** Functional cohesion is said to exist, if different elements of a module achieve a single function.

Classification of Coupling

- Coupling between two modules is a measure of the degree of interdependence or interaction between the two modules.
- A module having high cohesion and low coupling is said to be functionally independent of other modules.
- If two modules interchange large amounts of data, then they are highly interdependent.
- The degree of coupling between two modules depends on their interface complexity.
- The interface complexity is basically determined by the number of types of parameters that are interchanged while invoking the functions of the module.

Continue…

- The different classes of Coupling that can exist between module are shown in the following fig.

Data	Stamp	Control	Common	Content

Low (Best) $\longrightarrow$ high (Worst)

Fig. – Classification of coupling

Continue...

- **Data coupling**: Two modules are data coupled, if they communicate through a parameter. An example is an elementary data item passed as a parameter between two modules, e.g. an integer, a float, a character, etc.
- **Stamp coupling**: Two modules are stamp coupled, if they communicate using a composite data item such as a record in PASCAL or a structure in C.
- **Control coupling**: Control coupling exists between two modules, if data from one module is used to direct the order of instructions execution in another. An example of control coupling is a flag set in one module and tested in another module.

Continue…

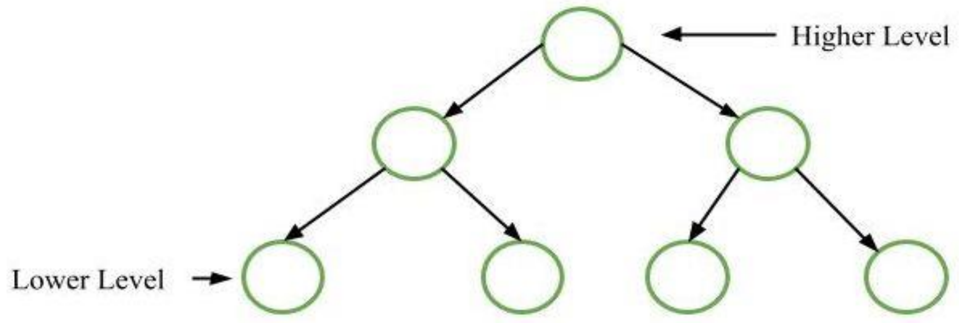
- **Common coupling:** Two modules are common coupled, if they share data through some global data items.

- **Content coupling:** Content coupling exists between two modules, if they share code, e.g. a branch from one module into another module.

- High coupling among modules not only makes a design difficult to understand and maintain but it also increase development effort because modules having high coupling can not be developed independently by different team members.
- modules having high coupling are difficult to implement and debug.

Function Oriented Software Design

- Function Oriented design is a method to software design where the model is decomposed into a set of interacting units or modules.
- Each unit or module has a clearly defined function.
- Thus, the system is designed from a functional viewpoint.



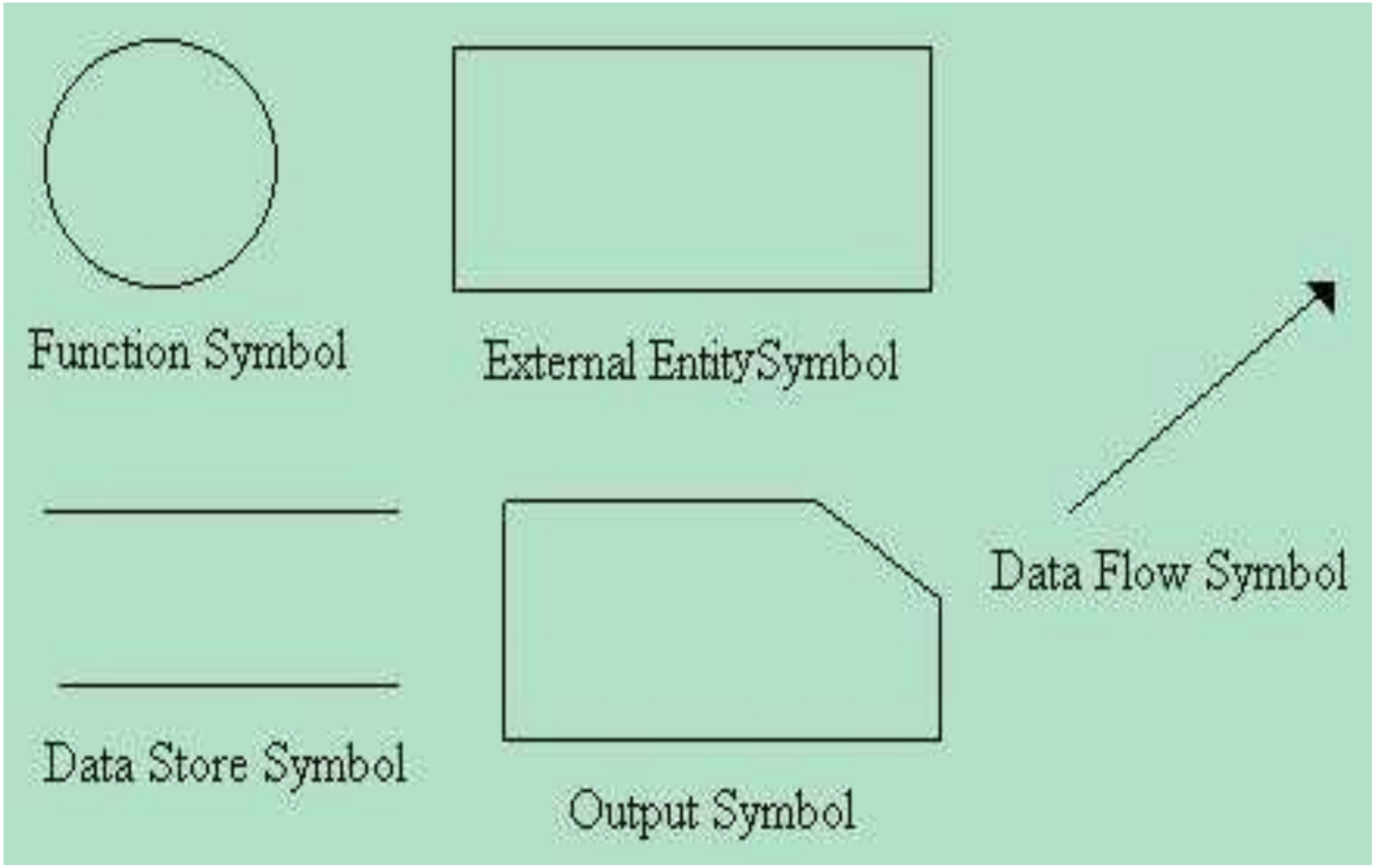
Data Flow Diagram (DFD)

- The DFD (also known as a bubble chart) 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.
- Each function is considered as a processing station (or 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.

Symbols used for designing DFDs

- Main five symbols used for constructing DFDs are:
- Function symbol
- External Entity Symbol
- Data Flow Symbol
- Data Store Symbol
- Output Symbol

Continue…


Function Symbol
External EntitySymbol
Data Flow Symbol
Data Store Symbol
Output Symbol

Continue…

- Function symbol is used to represent a function. This symbol is called a process or bubble. These bubbles are annotated with the corresponding function names.
- External Entity Symbol is used for representing those physical agents that are external to the software system. These agents interact with the software by providing or consuming data with the software system.
- Data Flow symbol shows the data flow between processes or physical agents and processes. The direction of data flow is given by the arrow and is usually labeled with corresponding data name.
- Data Store symbol represents a data structure or a logical file or a physical file on disk and will be connected to processes using data flow symbols.
- Output symbol is for representing output data produced by the software.

DFD Guidline

- The complete DFD of a system is not constructed in a single step. Rather, first an abstract DFD is constructed and is refined in further stages to obtain a detailed DFD.
- Construction of a DFD starts with the most abstract definition of the system.
- This abstract representation is called a Context Diagram or Level 0 DFD. In the Context Diagram the entire software is represented as a single bubble. The data inputs and outputs of the system is represented as a single bubble.
- The data inputs and outputs of the system is represented as incoming and outgoing arrows.
- In the Context Diagram the external entities with which the system interacts are identified and the data interchange occurring between the system and these external entities are represented.

Continue…

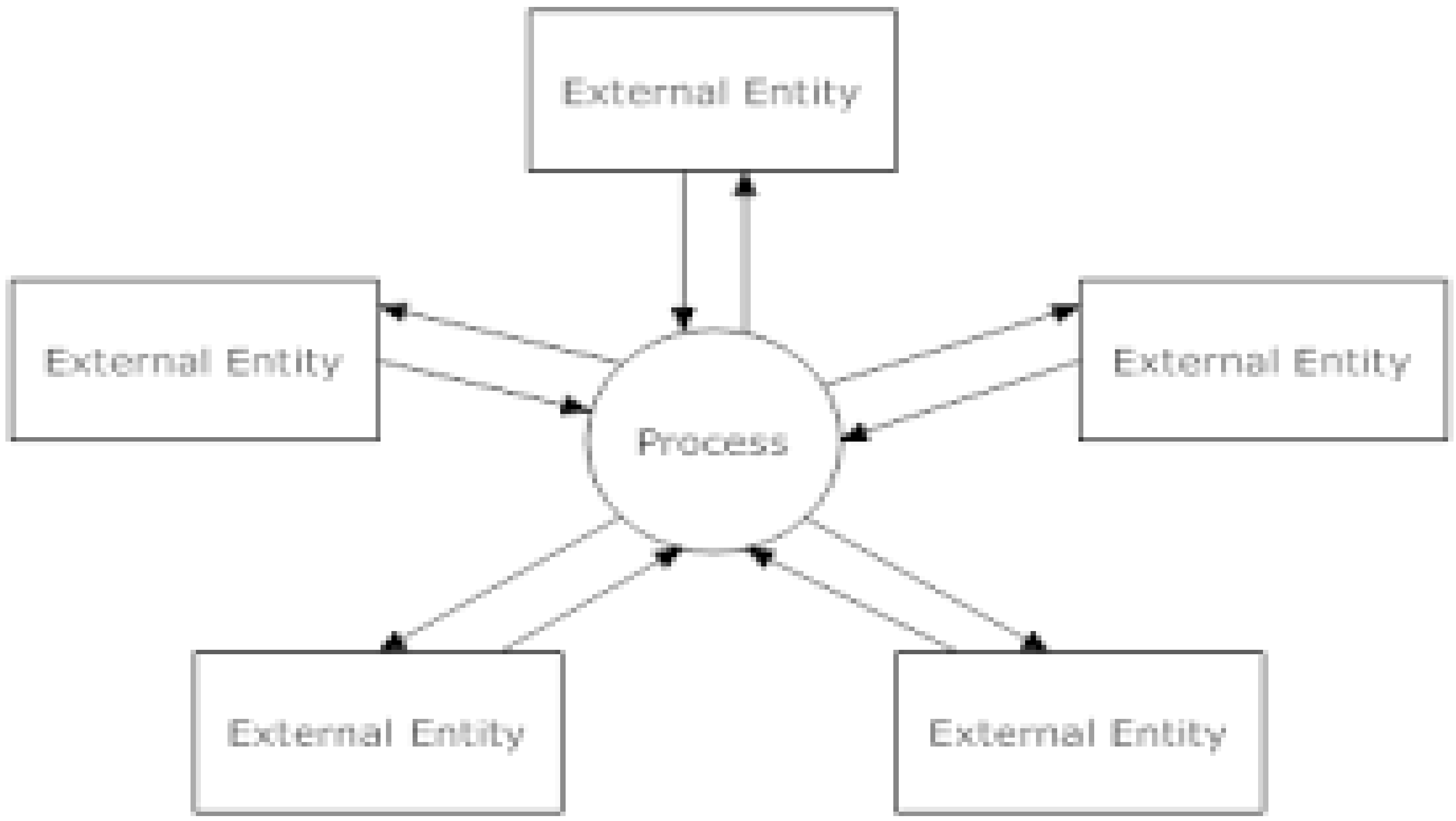
- **Numbering of Bubbles:-**
- It is necessary to number the different bubbles occurring in the DFD. These numbers help in uniquely identifying any bubble in the DFD by its bubble number.
- The bubble at the context level is usually assigned the number 0 to indicate that it is the 0 level DFD. Bubbles at level 1 are numbered, 0.1, 0.2, 0.3, etc, etc.
- When a bubble numbered x is decomposed, its children bubble are numbered x.1, x.2, x.3, etc.

Commonly made errors while constructing a DFD model

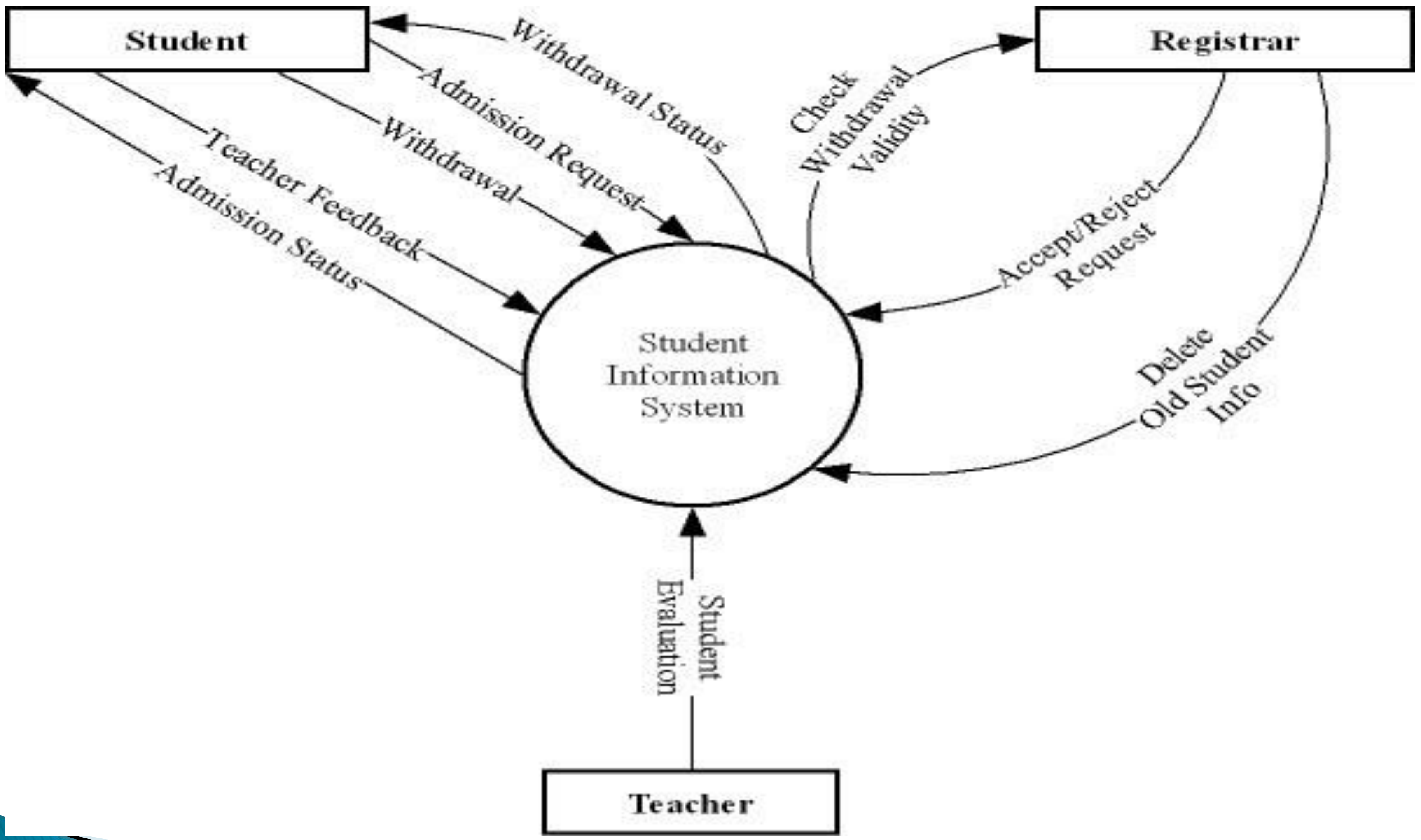
- Many beginners commit the mistake of drawing more than one bubble in the context diagram. A context diagram should depict the system as a single bubble.
- Many beginners have external entities appearing at all levels of DFDs. All external entities interacting with the system should be represented only in the context diagram. The external entities should not appear at other levels of the DFD.
- It is a common oversight to have either too less or too many bubbles in a DFD. Only 3 to 7 bubbles per diagram should be allowed, i.e. Each bubble should be decomposed to between 3 and 7 bubbles.
- Many beginners leave different levels of DFD unbalanced.

Continue…

- A common mistake committed by many beginners while developing a DFD model is attempting to represent control information in a DFD. It is important to realize that a DFD is the data flow representation of a system, and it does not represent control information.


External Entity
External Entity
External Entity
Process
External Entity
External Entity

CONTEXT LEVEL DIAGRAM



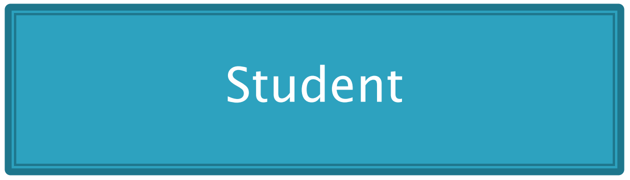
Data Modeling Concepts

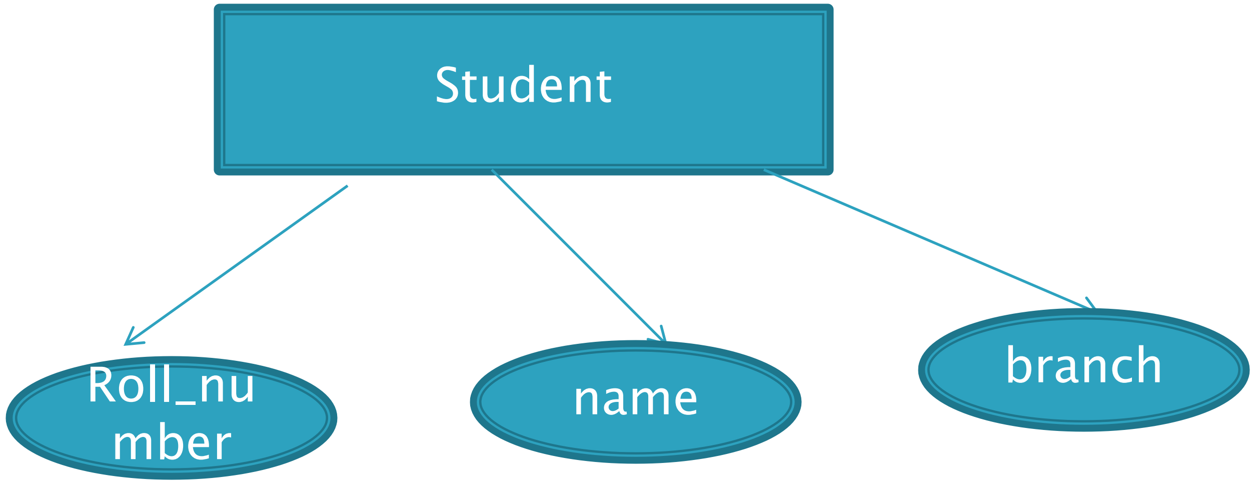
- Data Model is a conceptual relationship between data structure (tables) used in database.
- It provide abstract and conceptual representation of data.
- Data modeling or ER diagram gives the concept of Data objects (entity),attributes and relationship between objects .

Continue…

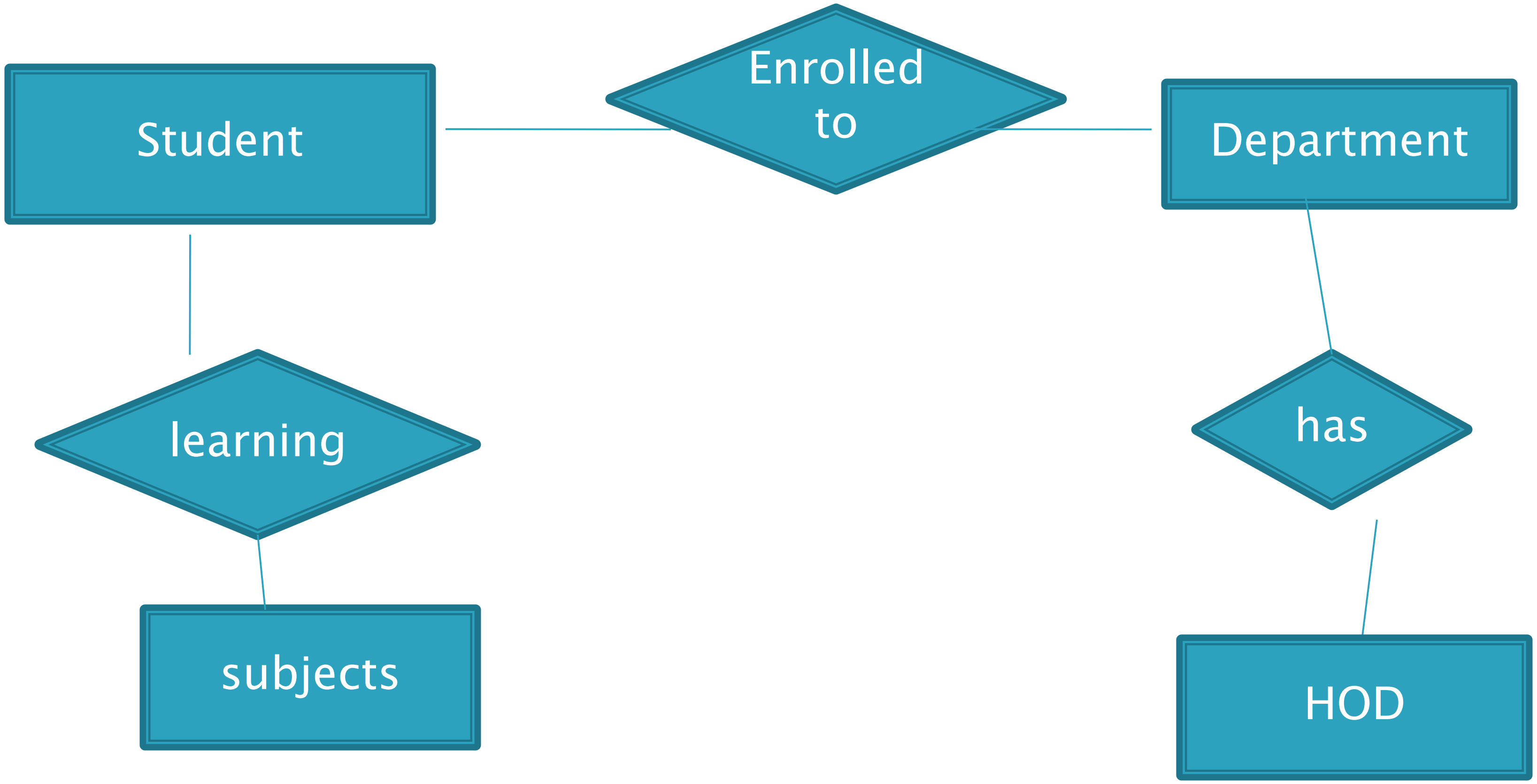
- Data Objects(Entity Set) : real world entity or thing for which you want to store data.
- Attributes : it is property of entity.
- Relationship : connection between entities.
- Cardinality : no of objects that participate in relationship. One to one ,one to many and many to many.
- Modality : Classification of relationship.it is 0 if no need for the relationship. And it is 1 if relationship is mandatory.

Entity and attribute

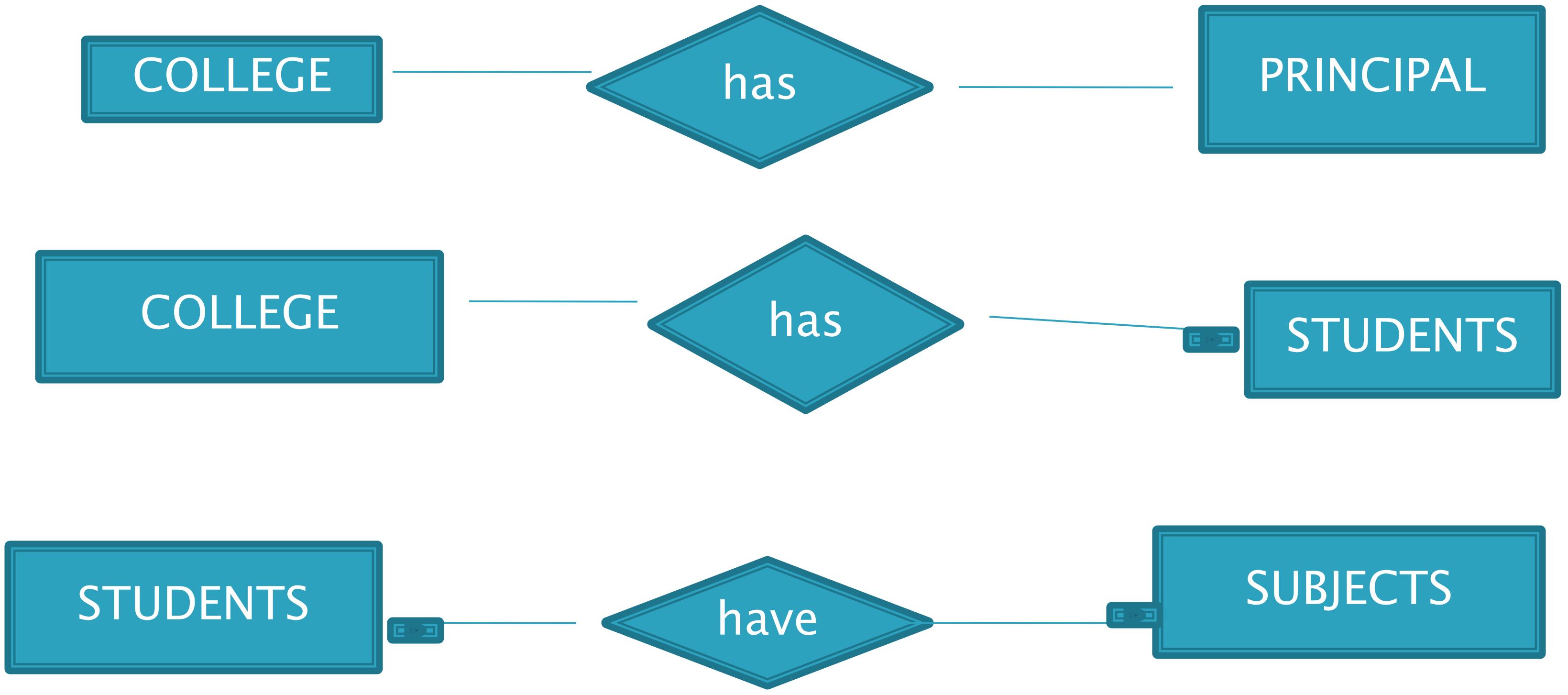


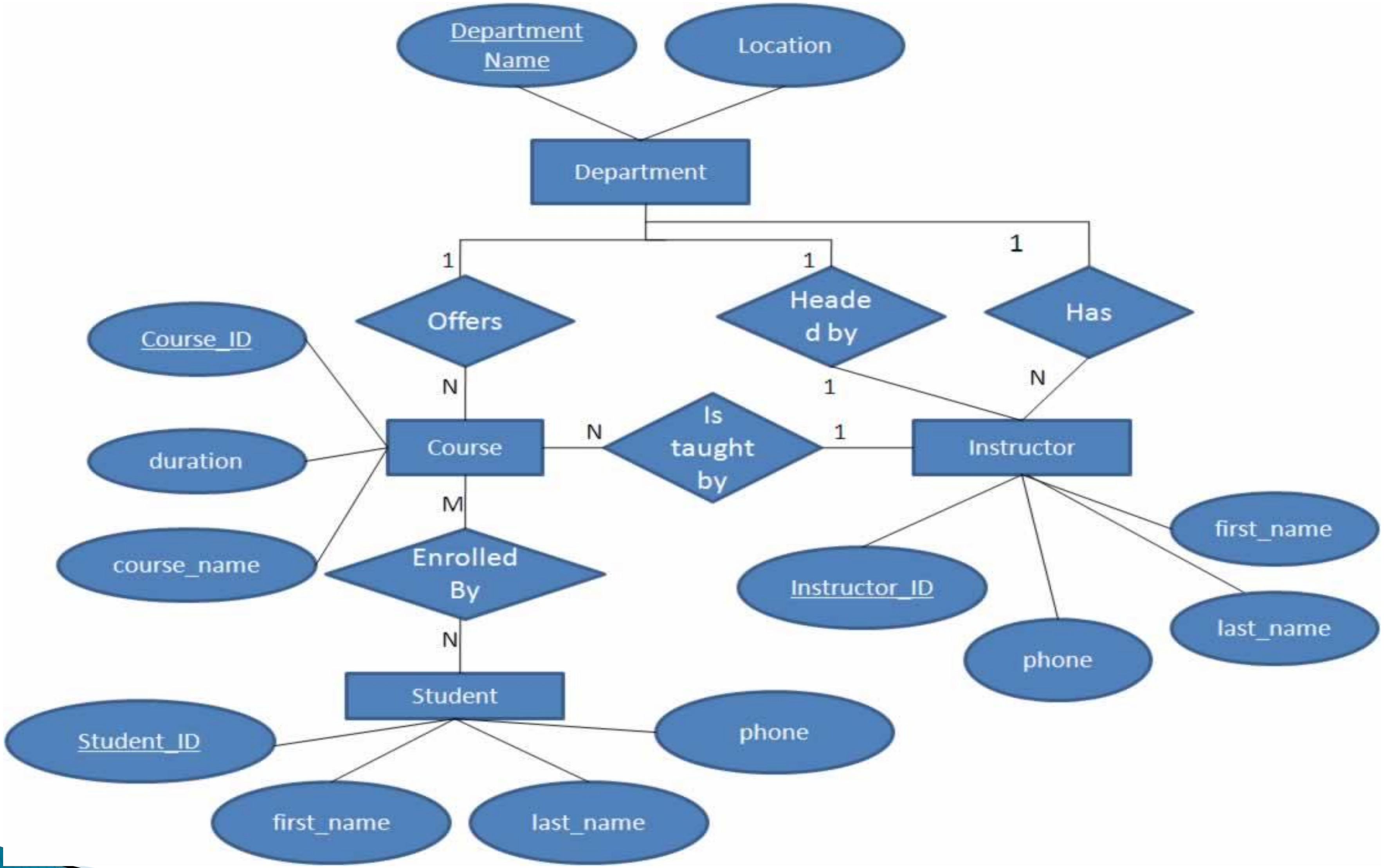


Relationship of entities.



Cardinality




Department Name
Location
Department
1
1
1
Offers
Heade d by
Has
Course_ID
N
1
N
duration
Course
N
Is taught by
1
Instructor
course_name
M
Enrolled By
first_name
Instructor_ID
last_name
phone
N
Student
Student_ID
phone
first_name
last_name

Object Modeling with UML

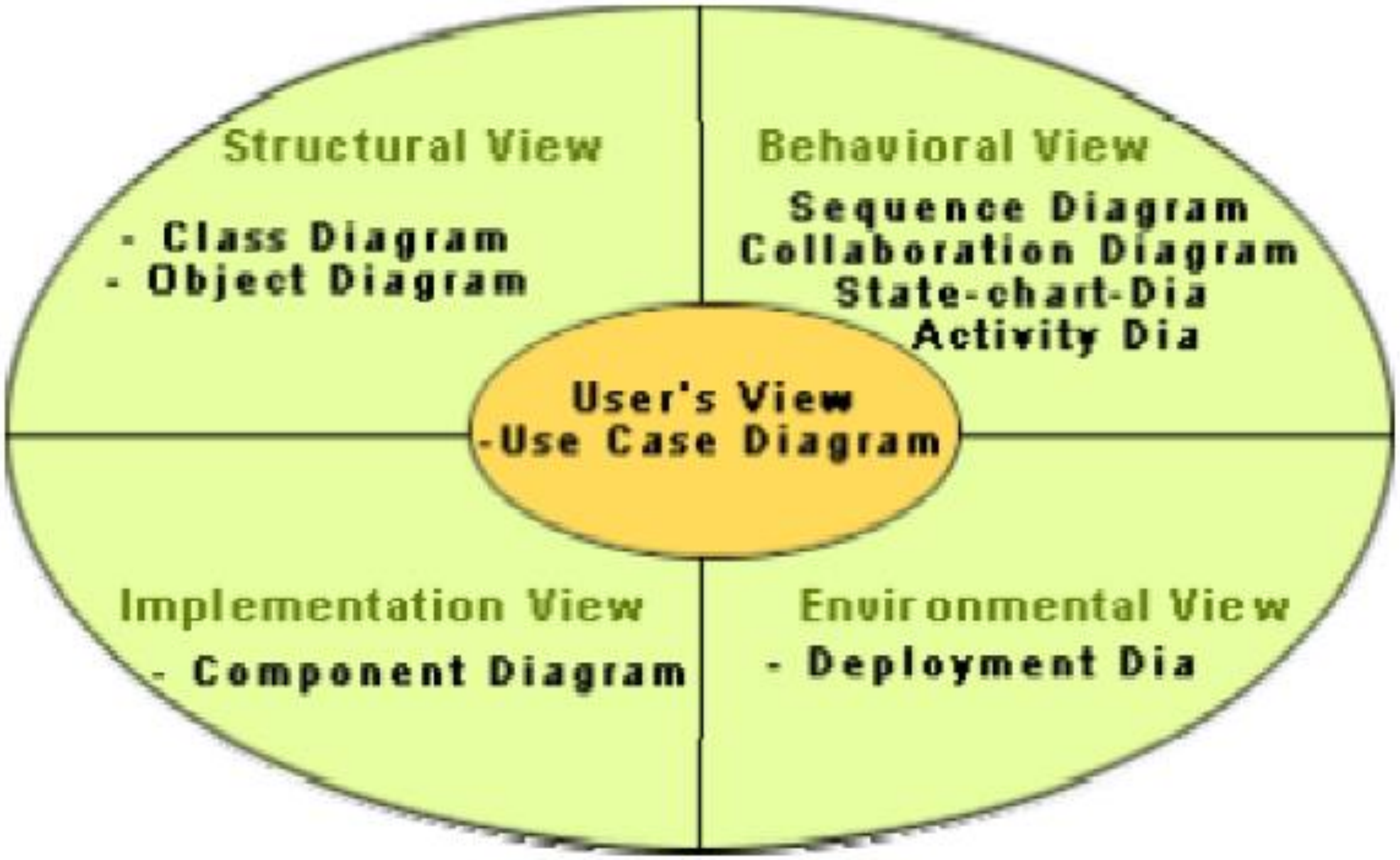
UML (Unified modeling language)

- UML is a modeling language.
- Standard language for specifying, visualizing, constructing, and documenting the software systems.
- The UML uses mostly graphical notations to express the design of software projects.
- It provides a set of notations (e.g. rectangles, lines, ellipses, etc.) to create a visual model of the system.
- UML Particularly useful for OO design.
- UML is Independent of implementation language

UML Diagrams

- UML can be used to construct nine different types of diagrams to capture five different views of a system.
- The UML diagrams can capture the following five views of a system:
- User’s view
- Structural view
- Behavioral view
- Implementation view
- Environmental view

Different views and diagrams



Continue…

- **User's view:** This view defines the functionalities (facilities) made available by the system to its users.
- The users' view captures the external users' view of the system in terms of the functionalities offered by the system.
- The users' view is a black-box view of the system where the internal structure, the dynamic behavior of different system components, the implementation etc. are not visible.

Continue…

- **Structural view:** The structural view defines the class and object which are important to the understanding of the working of a system and its implementation.
- It also captures the relationships among the classes(objects).
- The structural model is also called the static model, since the structure of a system does not change with time.

- **Behavioral view:** The behavioral view captures how objects interact with each other to realize the system behavior.
- The system behavior captures the dynamic behavior of the system.

Continue…

- **Implementation view:** This view captures the important components of the system and their dependencies.
- **Environmental view:** This view models how the different components are implemented on different pieces of hardware.

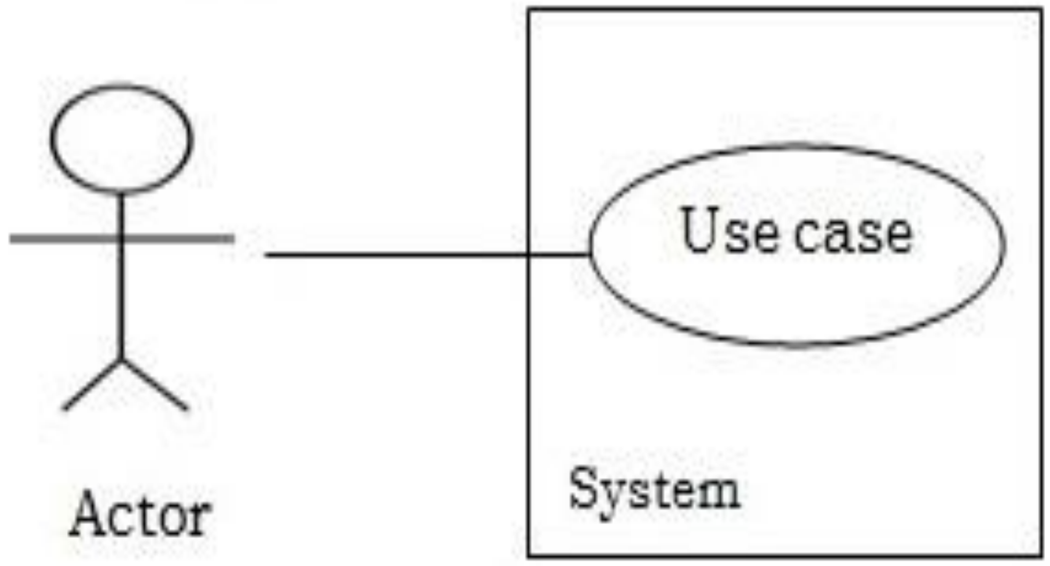
Use Case Diagram

- Use case model of the system consists of a set of use cases.
- Use cases represent the different ways in which a system can be used by the users.
- The purpose of a use case is to define the logical behavior of the system without knowing the internal structure of it.
- It provides understanding of the system.
- It identifies the functional requirements of the system.
- it describes "*who can do what in a system*".
- A use case typically represents a sequence of interactions between the user and the system.
- A simple way to find all the use cases of a system is to ask the question:"What the users can do using the system?"

Continue…

- Thus for the Library Information System (LIS), the use cases could be:
- • issue-book
- • query-book
- • return-book
- • create-member
- • add-book, etc

Continue…



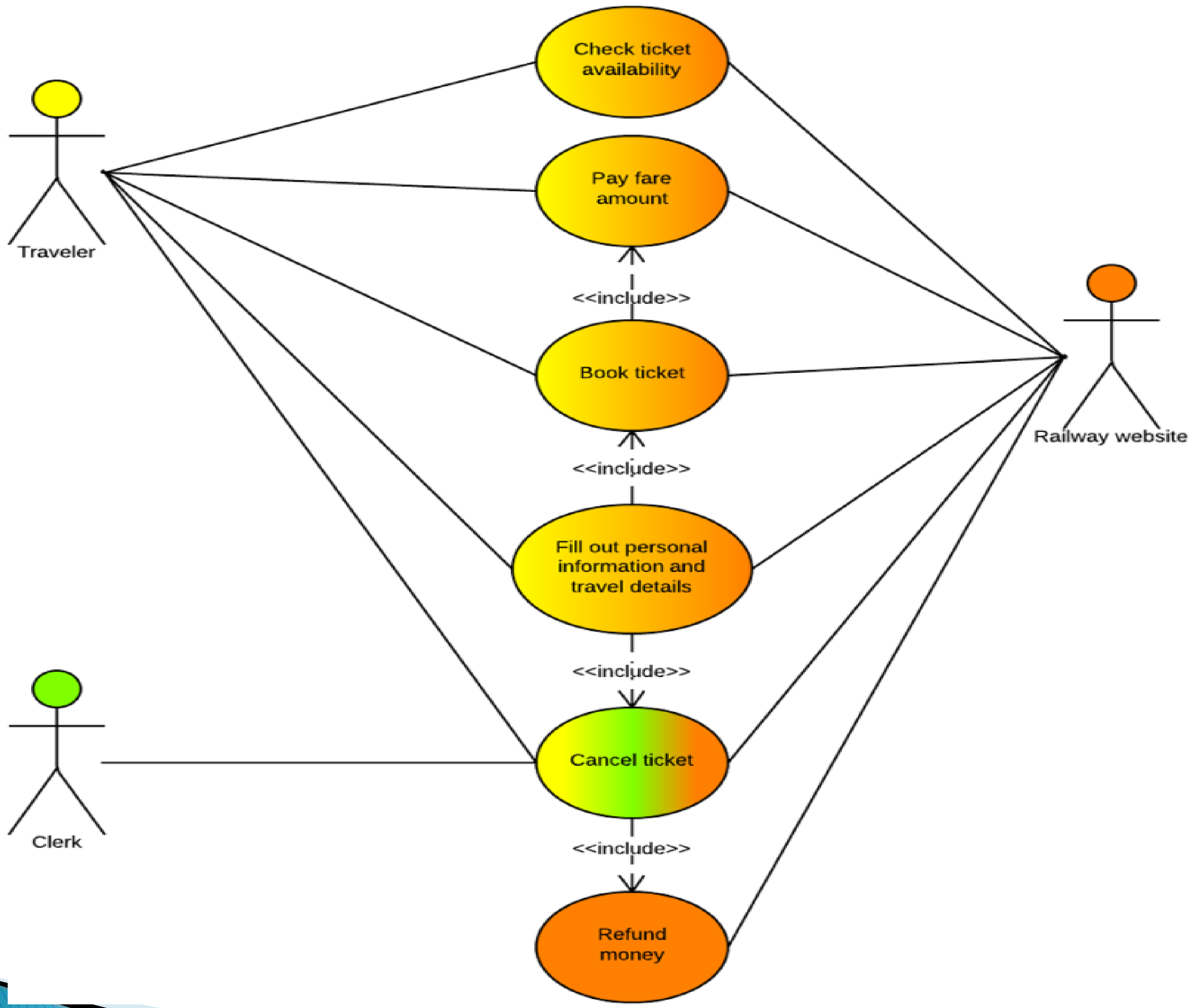
Continue…

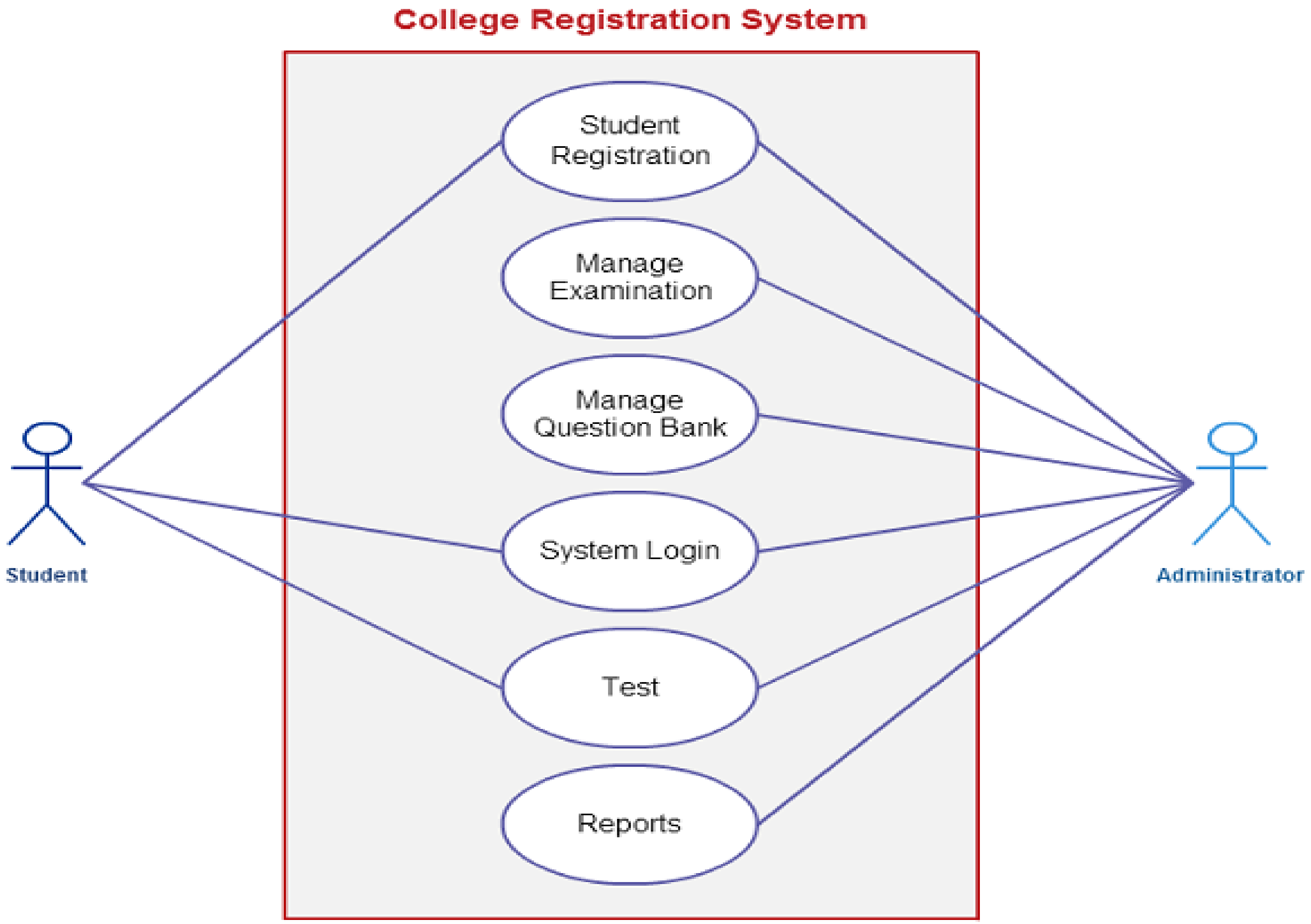
Use case:

- Represented by an ellipse with the name of the use case written inside the ellipse.
- All the use cases are enclosed with a rectangle representing system boundary. Rectangle contains the name of the system.
- *Actor:*
- An actor is anything outside the system that interacts with it.
- Actors are represented by using the stick person icon.
- An actor may be a person, machine or any external system.
- Actors are connected to use cases by drawing a simple line connected to it.

Continue…

- *Relationship:*
- It is also called communication relationship. Actors are connected to use cases through relationship lines.
- An actor may have relationship with more than one use case and one use case may relate to more than one actor.

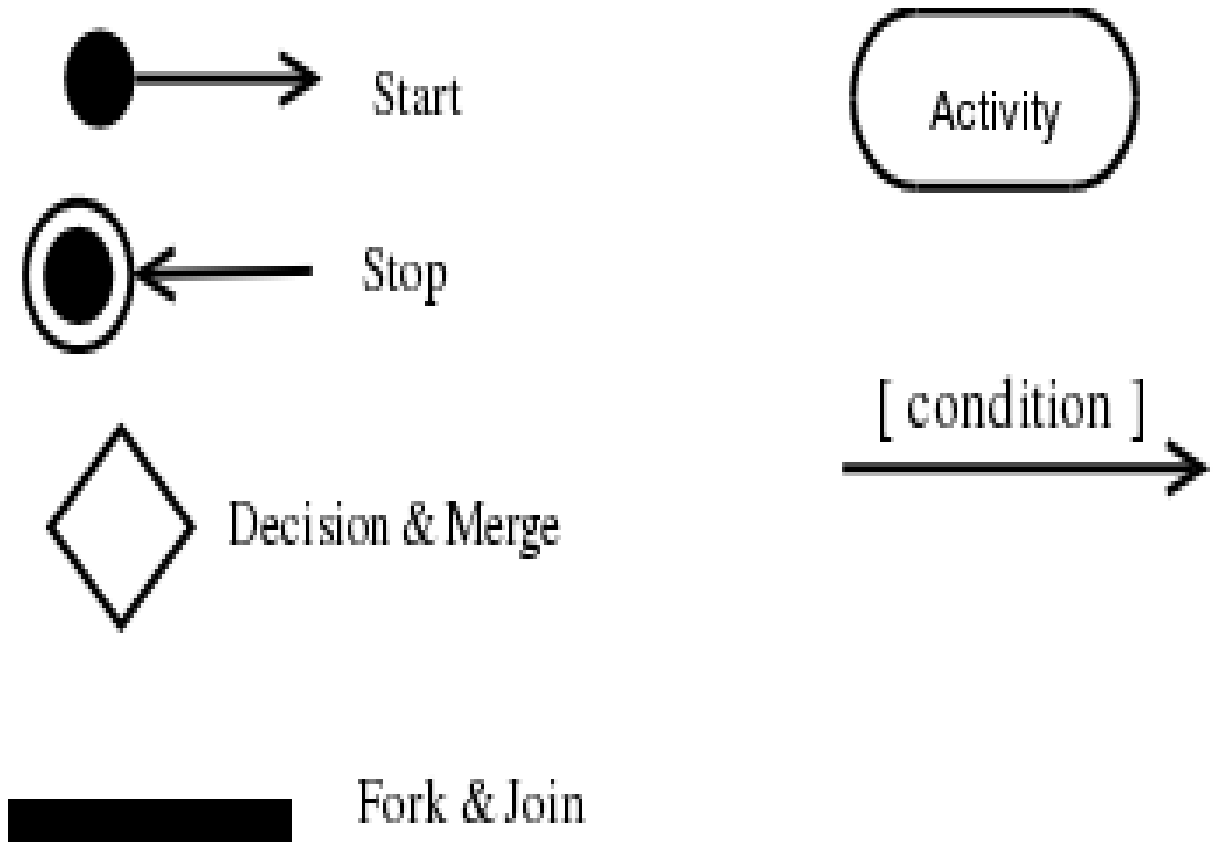

Check ticket availability
Pay fare amount
<<include>>
Book ticket
<<include>>
Fill out personal information and travel details
<<include>>
Cancel ticket
<<include>>
Refund money
Traveler
Clerk
Railway website

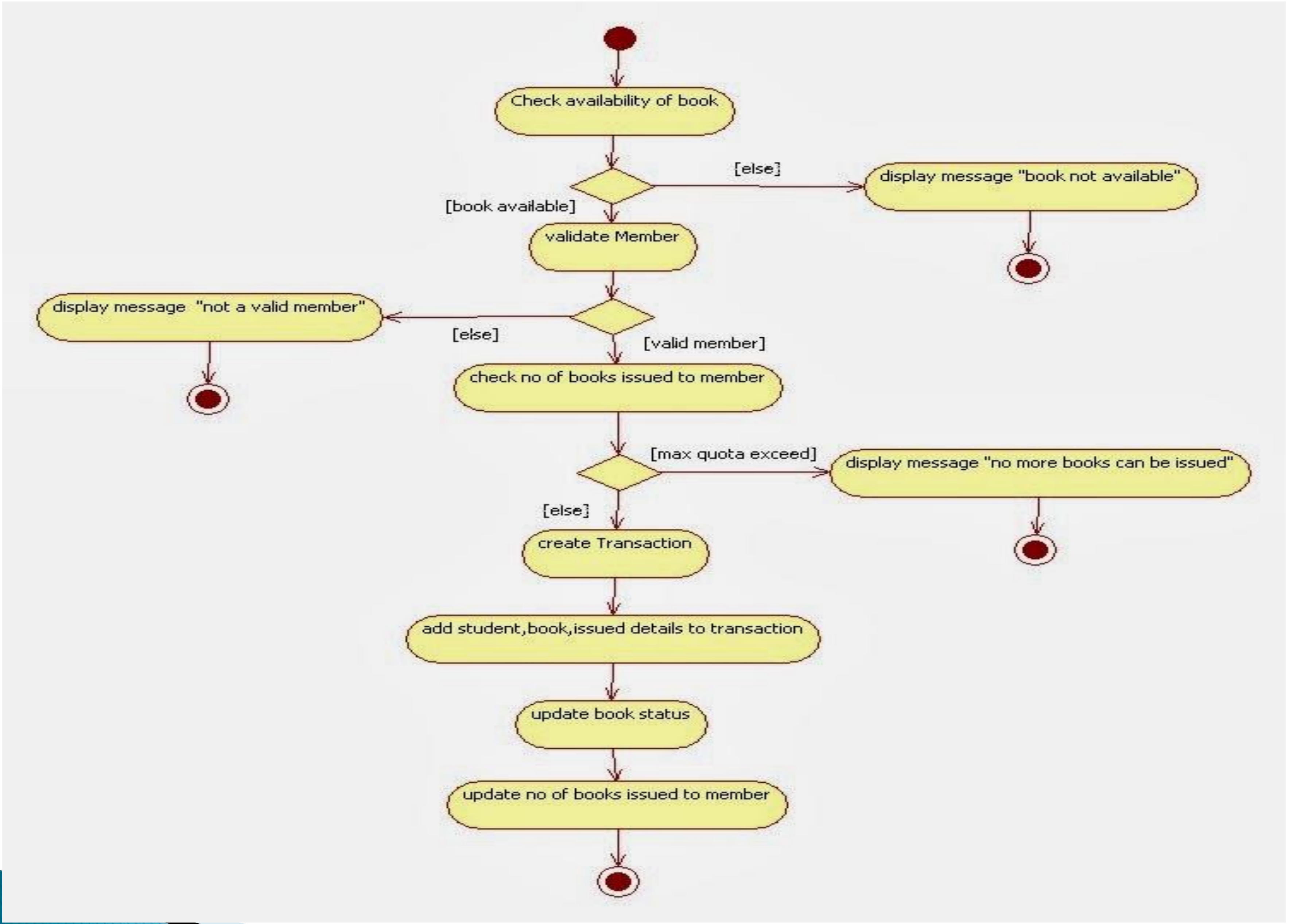

College Registration System
Student Registration
Manage Examination
Manage Question Bank
System Login
Test
Reports
Student
Administrator

Activity Diagram

- Activity diagrams represent workflows in an graphical way.
- The activity diagram focuses on representing activities.
- Activity diagrams are similar to 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 **swim lanes**. It enable you to group activities based on who performing them.
- Swim lanes subdivide activities based on responsibilities.

Continue...




Check availability of book
[else]
display message "book not available"
[book available]
validate Member
display message "not a valid member"
[else]
[valid member]
check no of books issued to member
[max quota exceed]
display message "no more books can be issued"
[else]
create Transaction
add student,book,issued details to transaction
update book status
update no of books issued to member

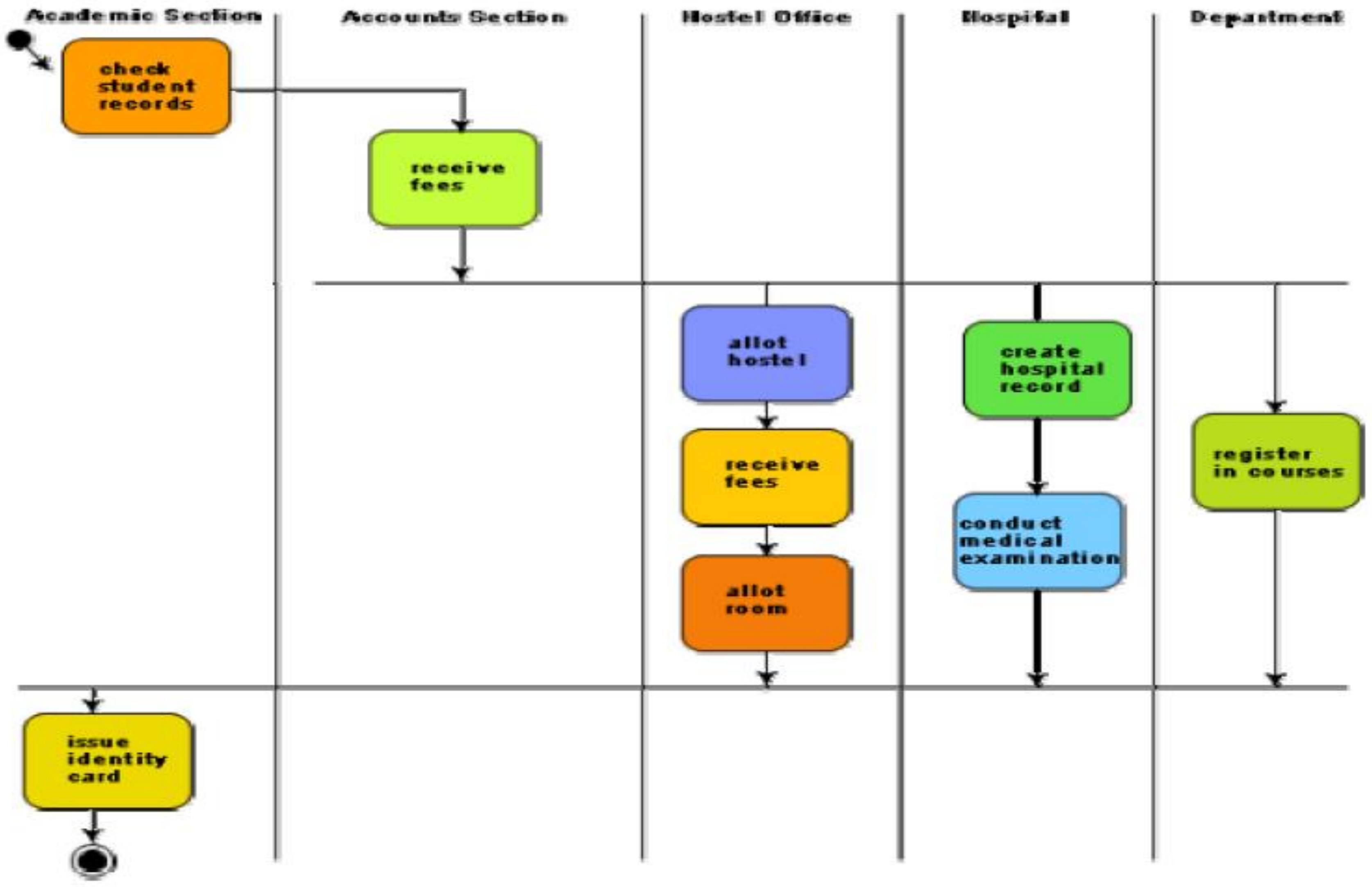


Fig. 7.15: Activity diagram for student admission procedure at IIT

Class Diagram

- The UML Class diagram is a graphical notation used to construct and visualize object oriented systems.
- A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's:
- classes,
- their attributes,
- operations (or methods),
- and the relationships among objects.

Continued...

UML class diagram is made up of

- set of classes and
- set of relationships between classes

What is a Class

- A description of a group of objects all with similar roles in the system, which consists of:
- **Structural features** (attributes) define what objects of the class "know"
  - Represent the state of an object of the class
  - Are descriptions of the structural or static features of a class
- **Behavioral features** (operations) define what objects of the class "can do"
  - Define the way in which objects may interact
  - Operations are descriptions of behavioral or dynamic features of a class

Continued…

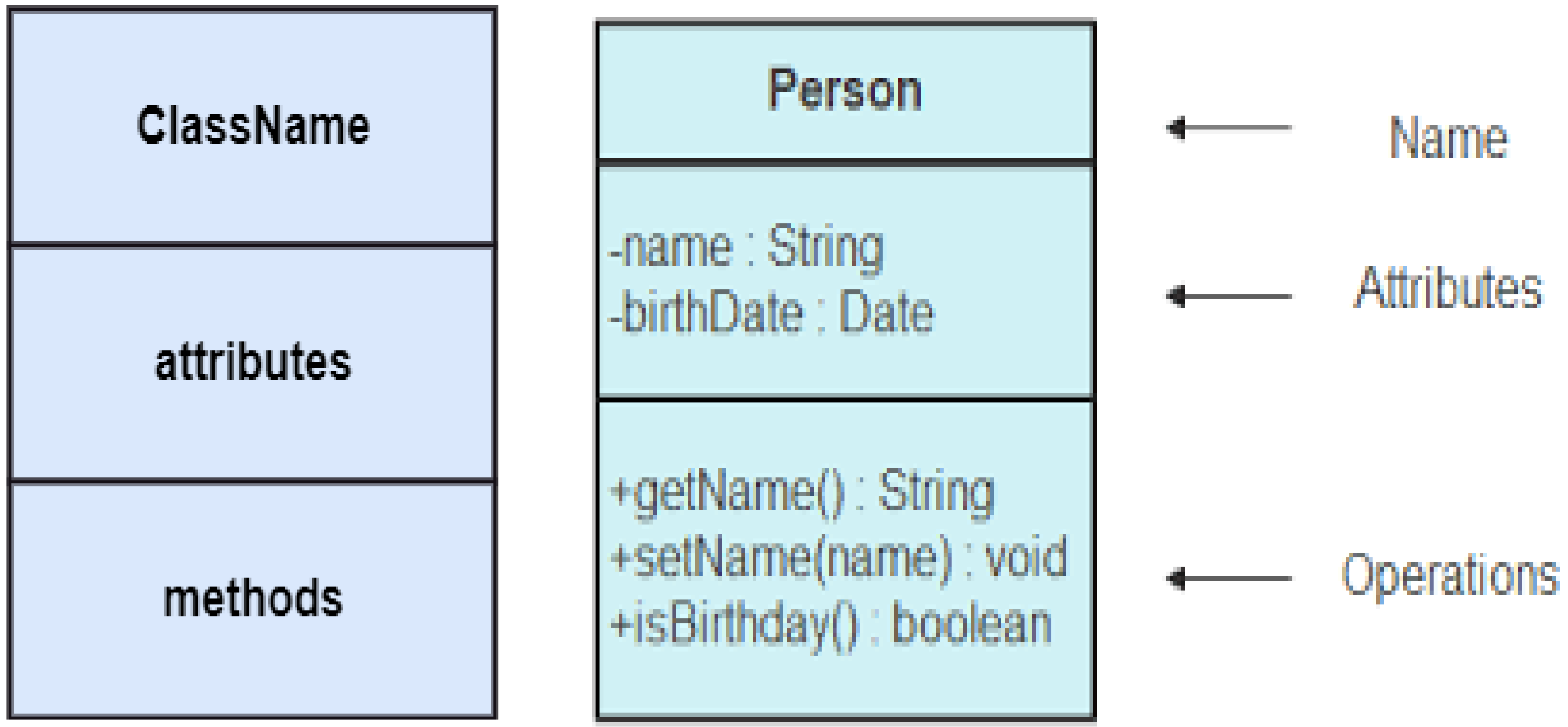
Class diagrams can be used for the following purposes

- To describe the static view of a system.
- To show the collaboration among every instance in the static view.
- To describe the functionalities performed by the system.
- To construct the software application using object-oriented languages.

Class Notation

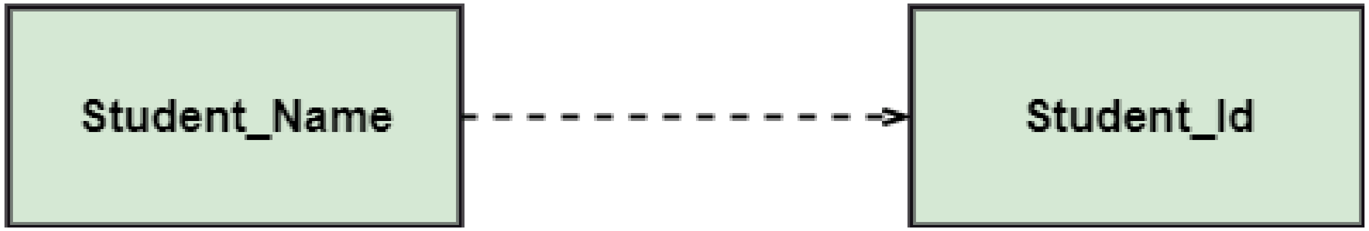
- A class notation consists of three parts:
- **Class Name**
  - The name of the class appears in the first partition.
- **Class Attributes**
  - Attributes are shown in the second partition.
  - The attribute type is shown after the colon.
  - Attributes map onto member variables (data members) in code.
- **Class Operations** (Methods)
  - Operations are shown in the third partition. They are services the class provides.
  - The return type of a method is shown after the colon at the end of the method signature.
  - The return type of method parameters is shown after the colon following the parameter name.
  - Operations map onto class methods in code

Continued…

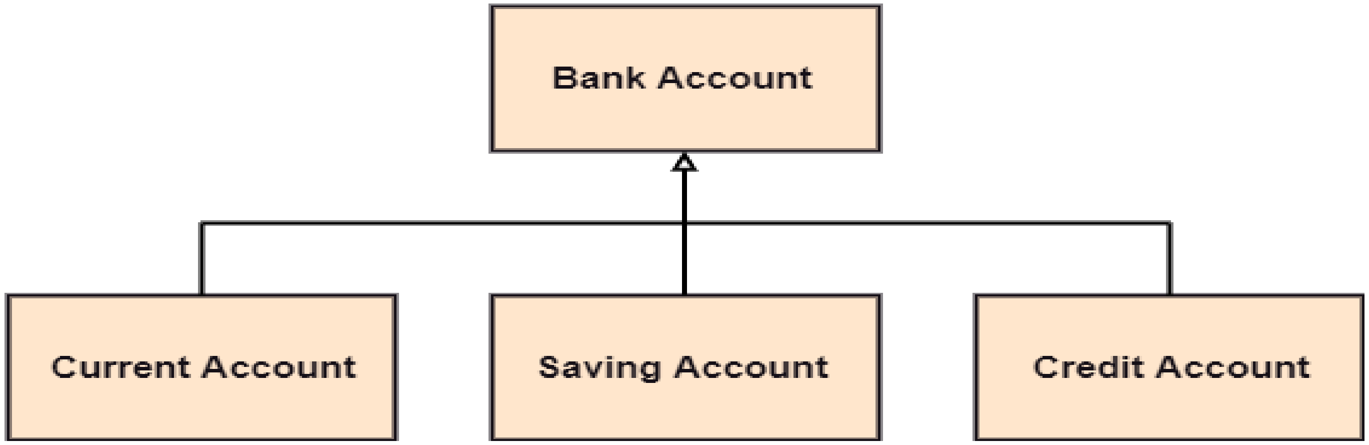

ClassName
attributes
methods
Person
-name : String
-birthDate : Date
+getName() : String
+setName(name) : void
+isBirthday() : boolean
Name
Attributes
Operations

Class Relationships

Dependency

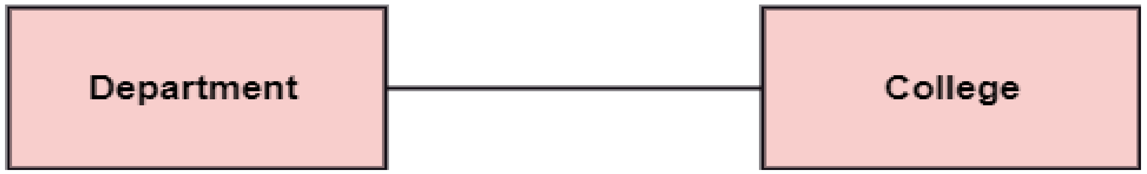


Generalization

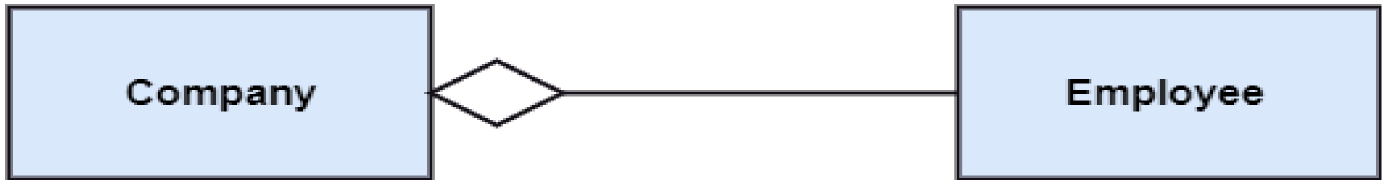


Continued...

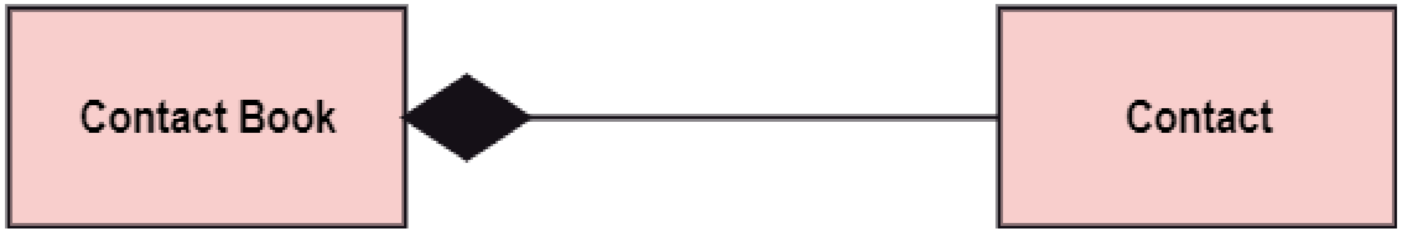
Association



Aggregation

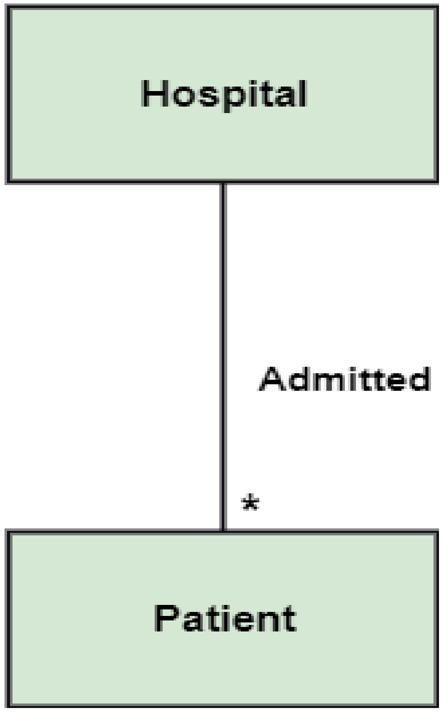


Composition

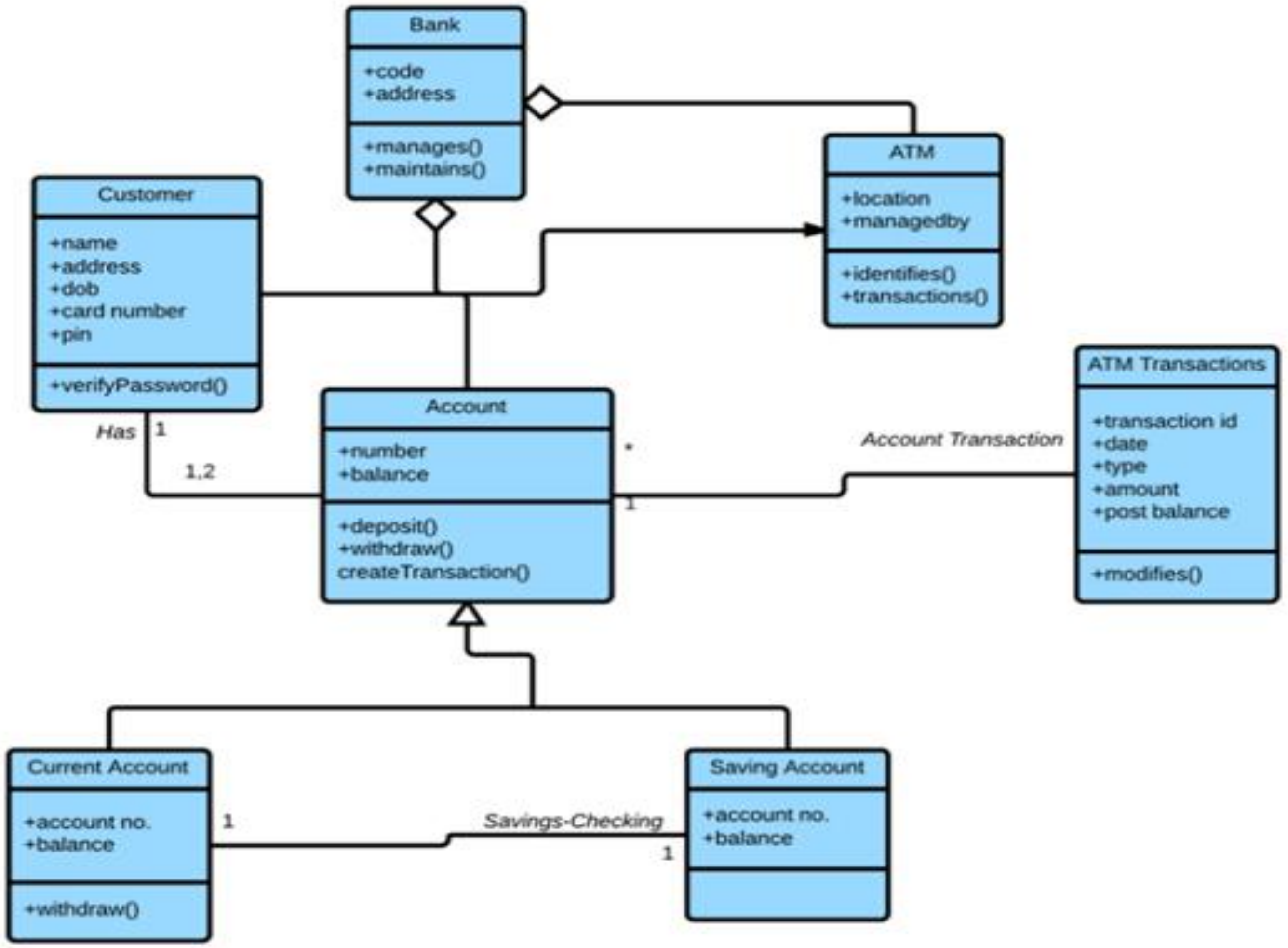


Continued…

Multiplicity



Example


Bank
+code
+address
+manages()
+maintains()
ATM
+location
+managedby
+identifies()
+transactions()
Customer
+name
+address
+dob
+card number
+pin
+verifyPassword()
Has
1
1,2
Account
+number
+balance
+deposit()
+withdraw()
createTransaction()
*
1
Account Transaction
ATM Transactions
+transaction id
+date
+type
+amount
+post balance
+modifies()
Current Account
+account no.
+balance
+withdraw()
1
Savings-Checking
1
Saving Account
+account no.
+balance

Sequence Diagram

- Sequence diagrams, commonly used by developers, model the interactions between objects in a single use case.
- They illustrate how the different parts of a system interact with each other to carry out a function, and the order in which the interactions occur when a particular use case is executed.
- In simpler words, a sequence diagram shows **how different parts of a system work in a ‘sequence’ to get something done.**

Continued…

- They can be used to model both simple and complex interactions between objects, making them a useful tool for software architects, designers, and developers.
- A **sequence diagram** is the most commonly used **interaction** diagram. **Interaction diagram** – An interaction diagram is used to show the **interactive behavior** of a system.
- A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram.

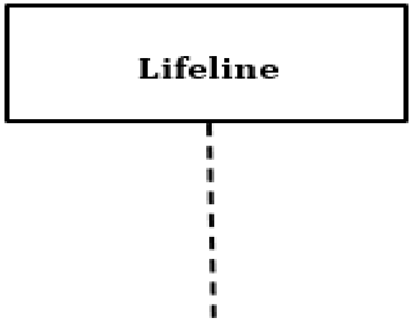
Purpose of sequence diagram

- To model high level interactions among active objects.
- Used to visualize the dynamic behaviour of the system.

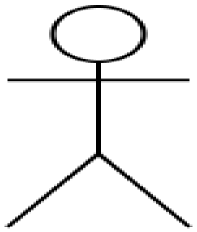
Components (notations) of sequence diagram

- **Lifeline**
- It is individual participant.
- Located at the top of the diagram.
- Each lifeline is represented by a vertical dashed line that extends down the length of the diagram and a rectangle box at the top of line is labelled with the name of the object or component it represents.

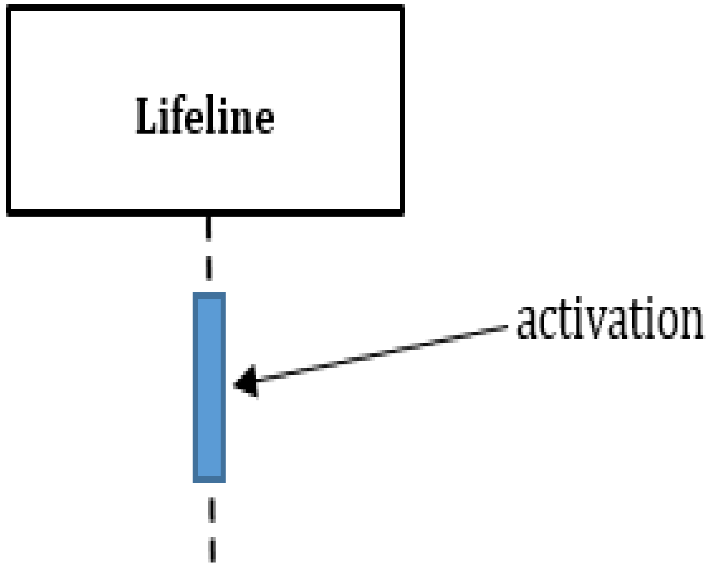
- It indicates the periods during which an object exists or is active.



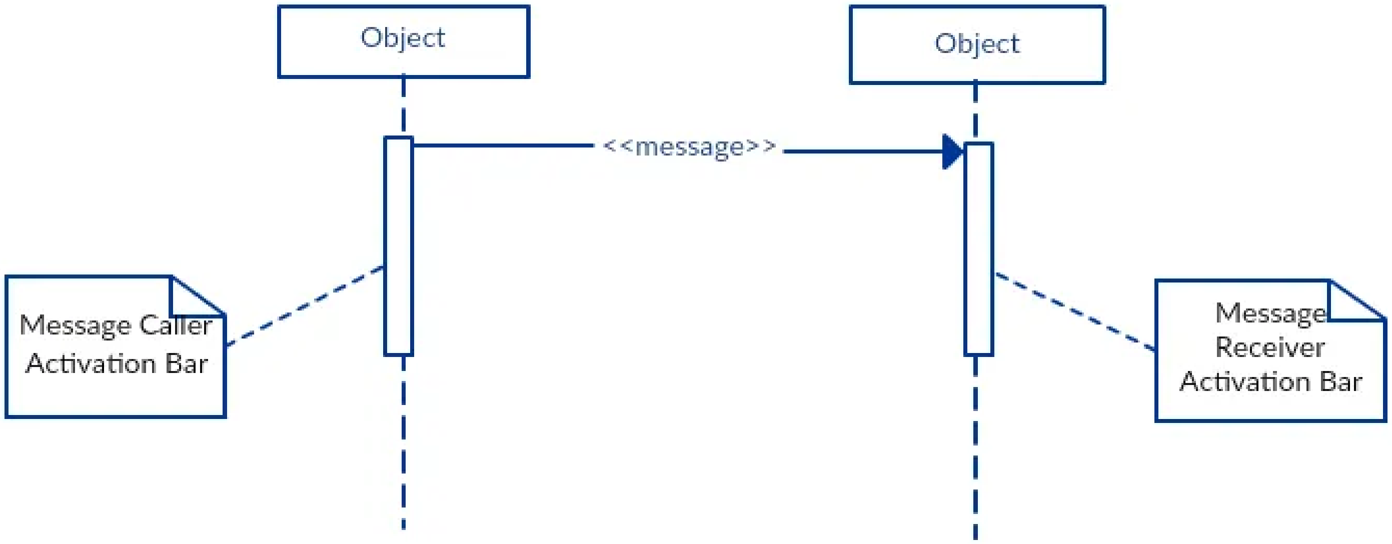
- **Actor**
- An actor represents a user, system, or external entity that interacts with the system.
- Actors are represented as stick person icon outside the system boundary, they are connected to the system by a dashec



- **Activation**
- It is runtime behaviour of the object.
- It is showing the time of object where it is activated for particular duration.
- It is represented by a thin rectangle on the lifeline.

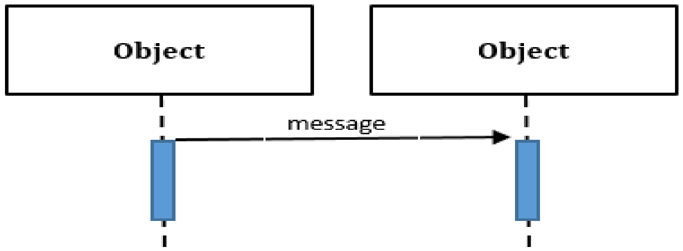


- **Message**
- Message represents the communication between objects.
- Used to convey the information from one object to another. (typically in case of request or response)
- Messages are represented by arrows that connect the lifelines of objects. The arrowhead indicates the direction of the message, and the label on the arrow indicates the type of message being sent.

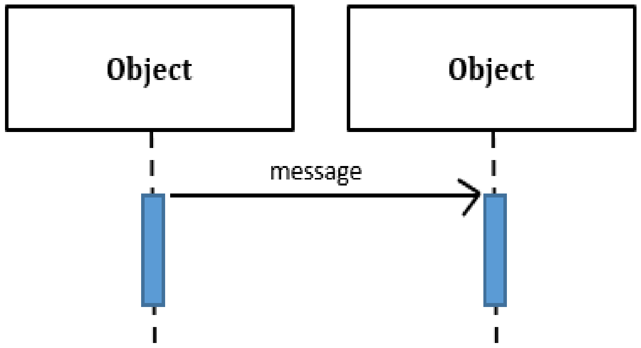

Object
Object
<<message>>
Message Caller Activation Bar
Message Receiver Activation Bar

Different types of messages used in class diagram

- **Call message**
- It defines a particular communication between Lifelines.
- Call message is of two types:
  - ***Synchronous call message***

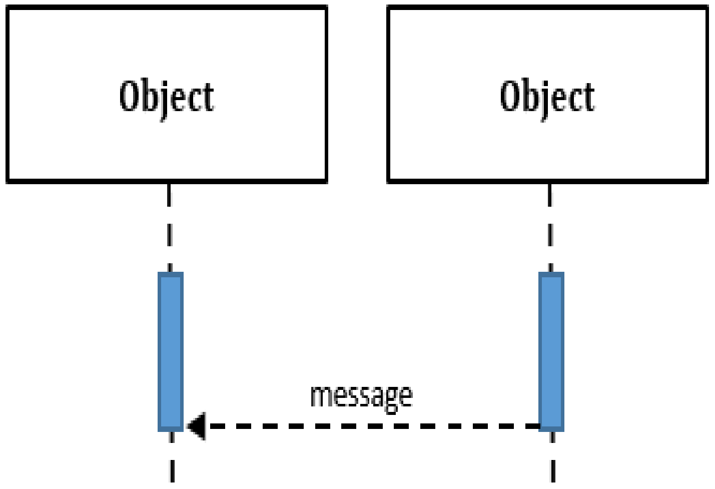


  - ***Asynchronous call message***



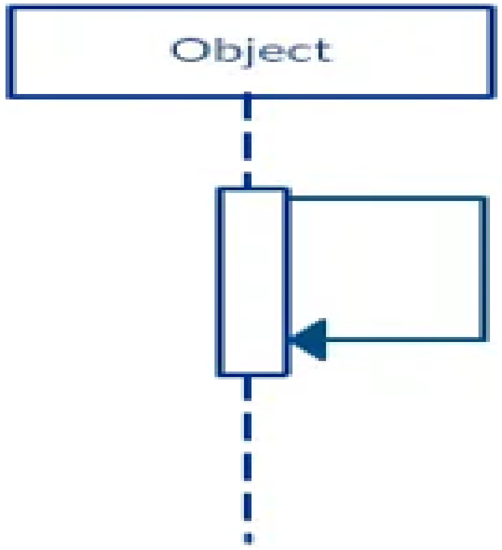
Return message

- A return message is used to return the result of an operation to the calling object.
- It is represented by dashed arrow.
- It is only used in response to synchronous messages.

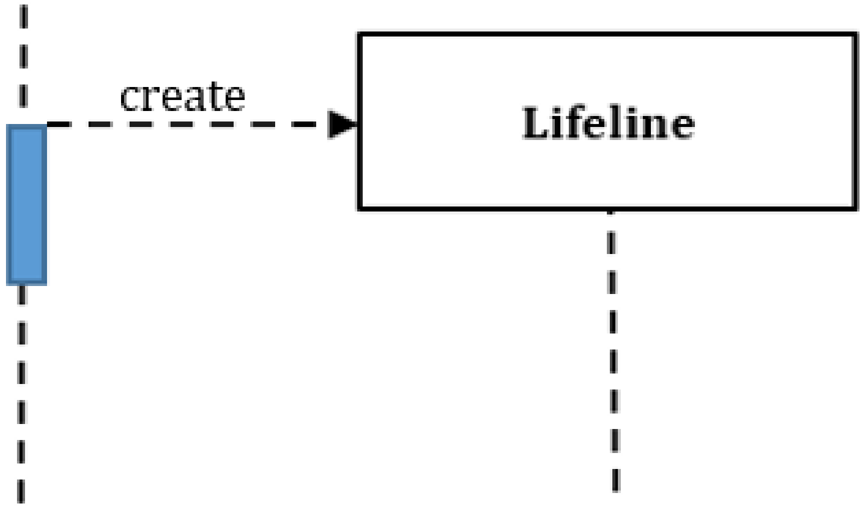


Self message

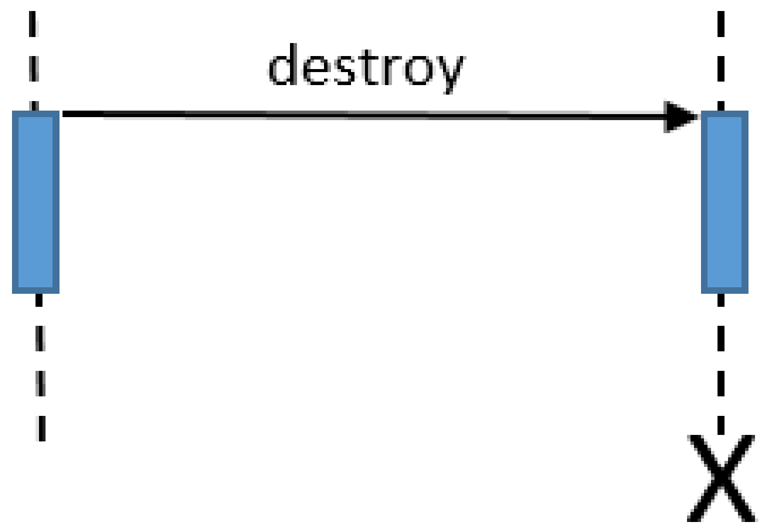
- A message from an object to itself.
- It is represented by a looped arrow.
- Self-messages are often used when an object needs to perform an internal operation or to trigger another operation within itself.



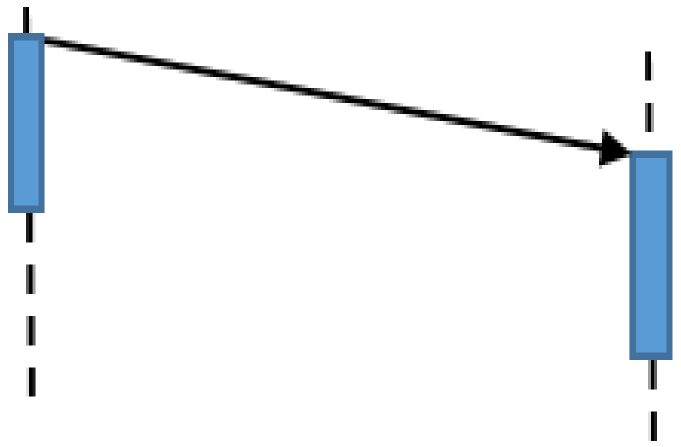
- **Create message**
- It describes creation of a new object or instance of a class.
- It is shown as a dashed line with open arrowhead (looks the same as reply message), and pointing to the created lifeline's head.



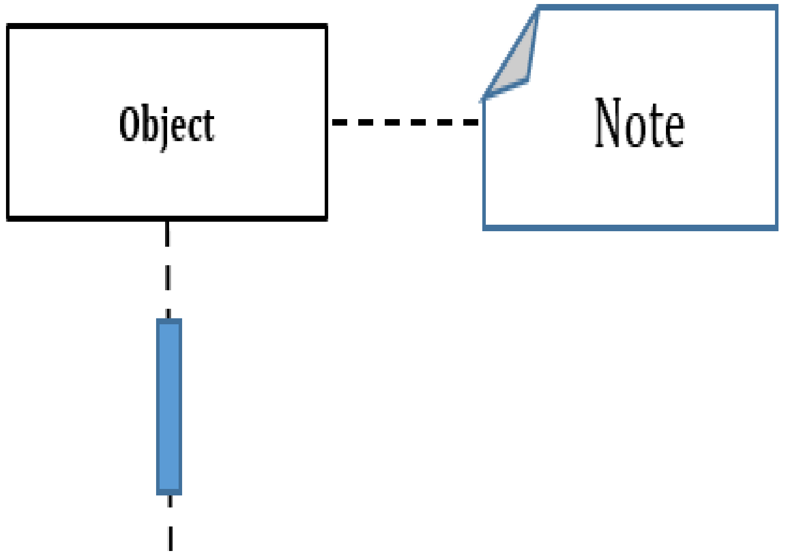
- **Destroy message**
- It represents the deletion or destruction of an object.
- It is represented by an arrow terminating with a X.



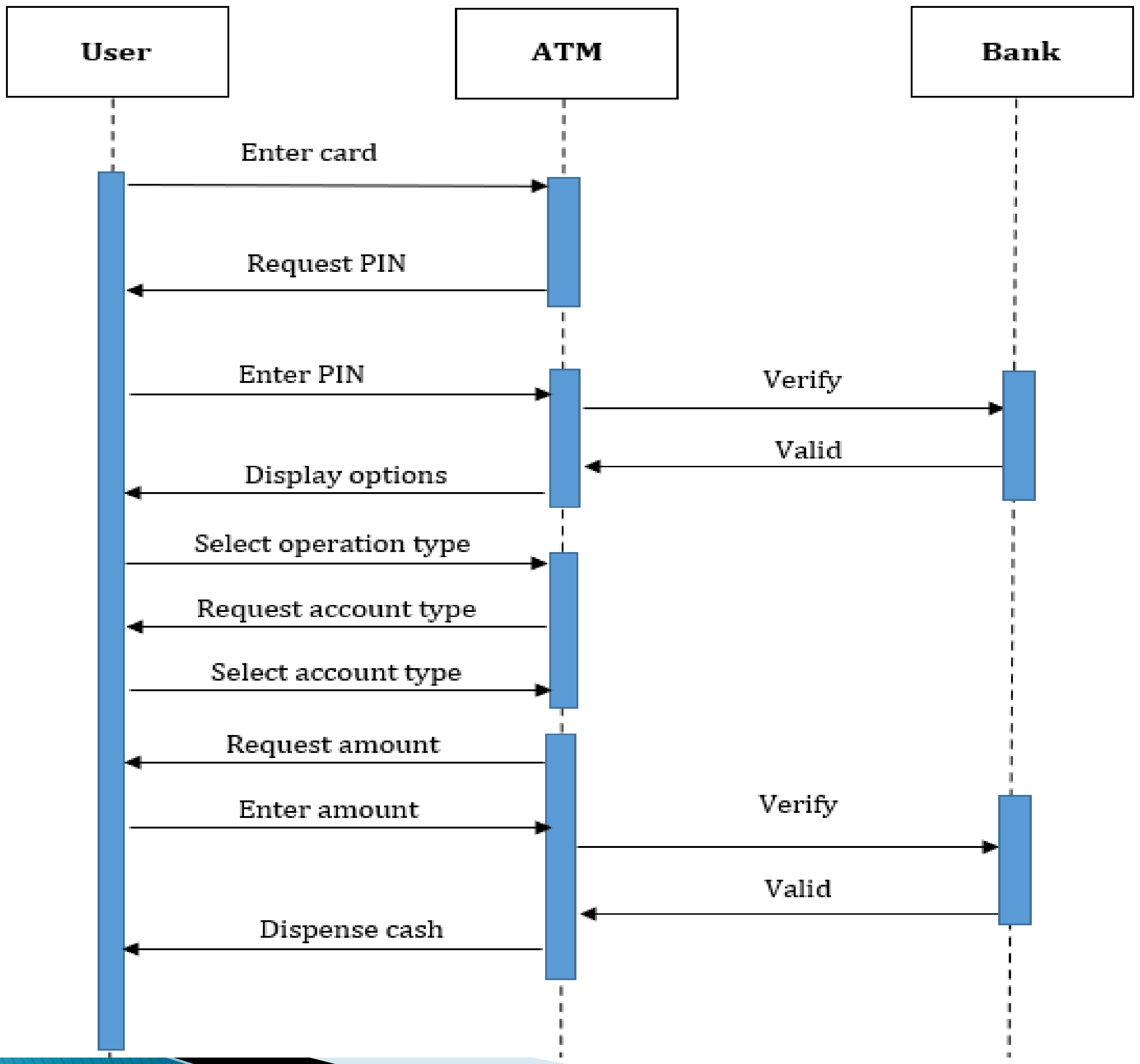
- **Duration message**
- That represents the duration of an activity in the system.
- It is denoted by a horizontal line with an optional label indicating the duration of the activity.



- **Note (Or Comment)**
- Provide remarks.
- Basically carries useful information.
- It is represented by a rectangle with a folded corner. And it can be linked with the related object with dashed line.



- Example of Sequence diagram (ATM system – '*withdraw amount*')


User
ATM
Bank
Enter card
Request PIN
Enter PIN
Verify
Valid
Display options
Select operation type
Request account type
Select account type
Request amount
Enter amount
Verify
Valid
Dispense cash