

Build COMPETENCY  
across your TEAM



**Microsoft Partner**  
Gold Cloud Platform  
Silver Learning

# UML And Modeling



# UML And Modelling

## Module 1: Overview of UML

# Why Modeling ?

- “A picture is worth a thousand words.” - old saying



# Principles of Modeling

- Add value to a “view” of the system .
- Reflects static as well as dynamic aspects.
- Shows different granularity or degrees of details.
- Choose your model





# What is UML?

- UML stands for Unified Modeling Language. It's an international industry standard graphical notation used for describing, visualizing, constructing and documenting the artifacts of a software system.
- UML - Unified Modeling language
- Blue print of source code.
- UML is a modeling language, not a methodology or process
- Developed by Grady Booch, James Rumbaugh and Ivar Jacobson at Rational Software and accepted as a standard by the Object Management Group (OMG), in 1997.





# Scope of UML

- **Programming Language:** UML is a “visual modeling language”. It is not intended to be a visual programming language. It is a language for visualizing, specifying, constructing, and documenting the artifacts of a software intensive system.
- **Tools:** Language standards form the foundation for tools and process. UML defines a semantic metamodel, and not a tool interface, storage, or run-time model.
- **Process:** Software development process will use UML as a common language for its project artifacts, but will use the same type of UML diagram in the context of different processes. UML is process independent

- Things

-  *Structural*
-  *Behavioral*
-  *Grouping*
-  *Annotational*

- Relationships

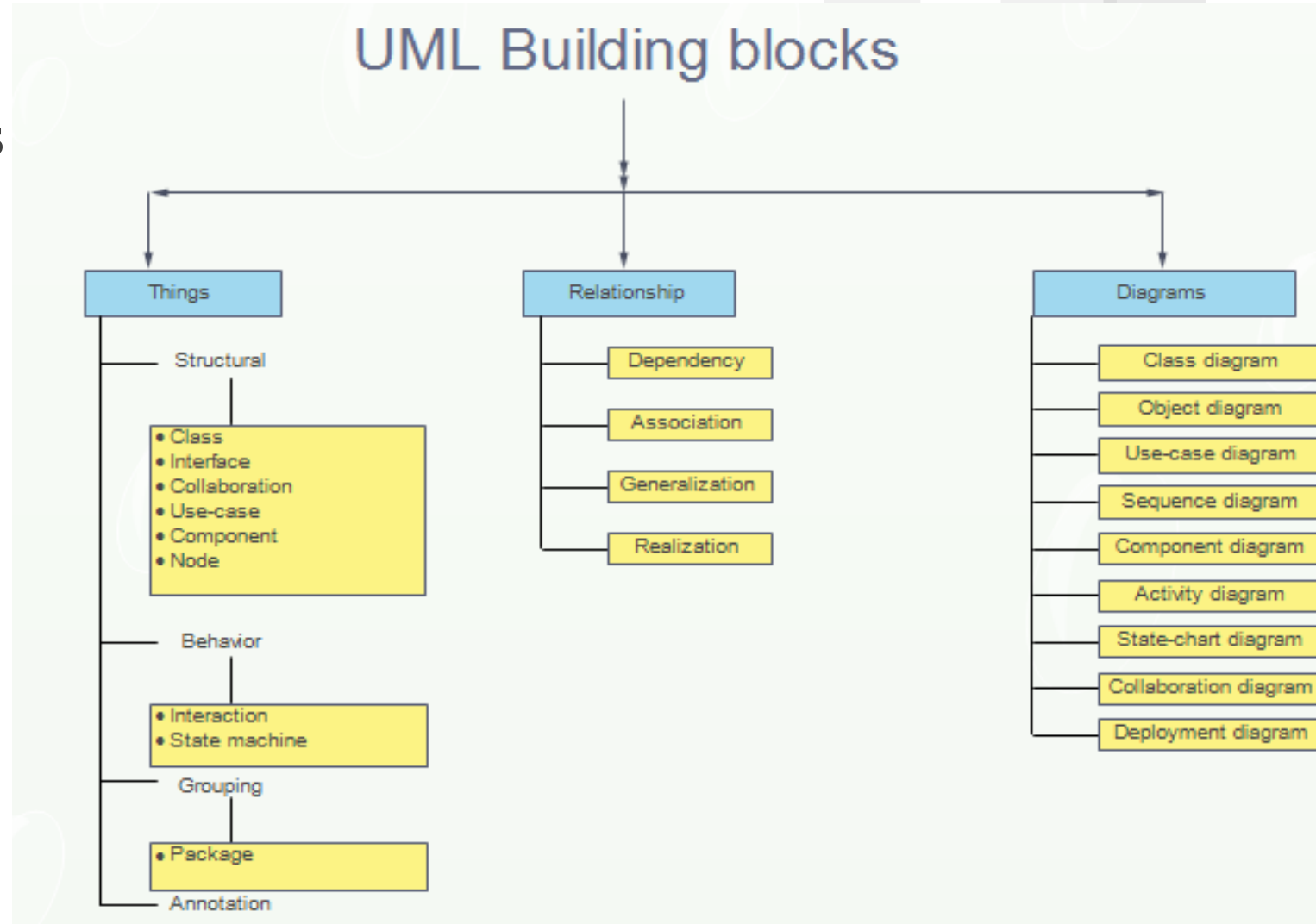
-  *Dependency*
-  *Association*
-  *Generalisation*
-  *Realization*

## ◆ Diagrams

- ◆ Class Diagram
- ◆ Object Diagram
- ◆ Use Case Diagram
- ◆ Sequence Diagram
- ◆ Collaboration Diagram
- ◆ Statechart Diagram
- ◆ Activity Diagram
- ◆ Component Diagram
- ◆ Deployment Diagram

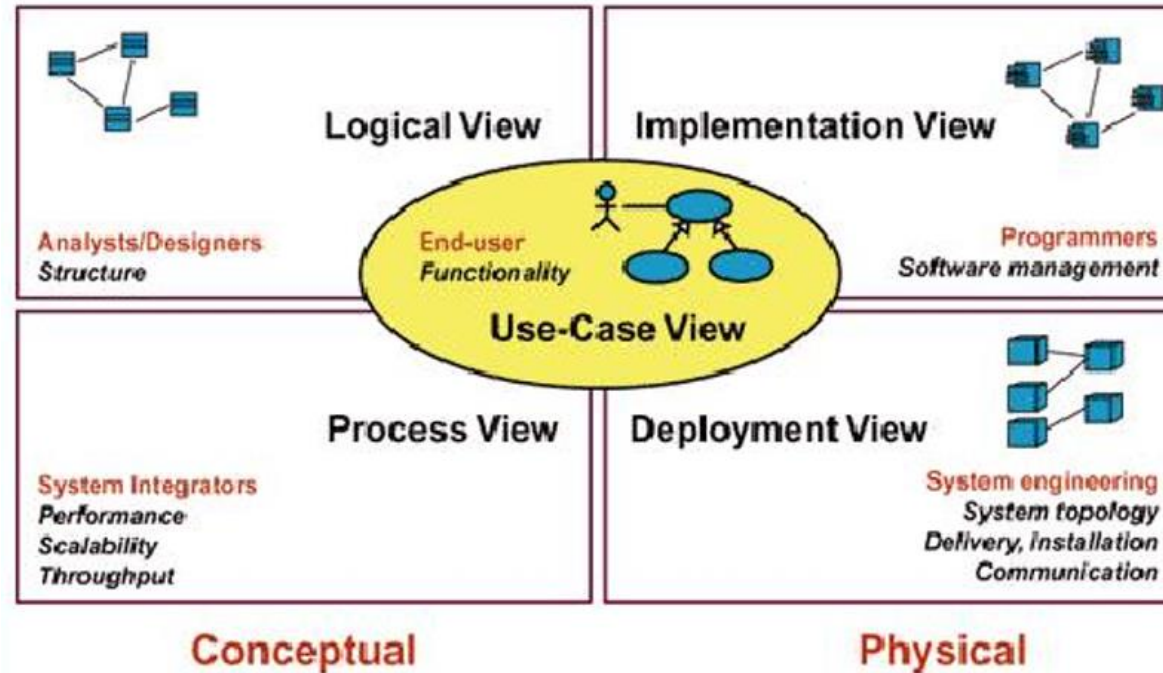
# Basic Building Blocks of UML Diagram (Select any one slide)

- Things
- Relationships
- Diagrams





# UML Building Blocks –Views



UML offers different views of the system, each **view containing different diagrams**.

The diagrams are made up of specific modelling elements.

In addition to existing modelling elements, UML allows extending available notation and semantics by the use of extension mechanisms

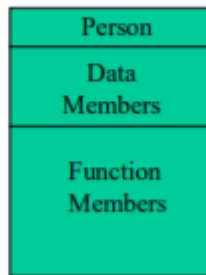
# Things

- **Structural things** -- The nouns of UML models. These represent elements that are conceptual or physical.
- **Behavioral things** -- Dynamic parts of UML models. The verbs which represent behavior over time and space.
- **Grouping things** -- Organizational parts of UML. These are boxes into which models can be decomposed.
- **Annotational things** -- Explanatory parts of UML. Used to describe, illuminate, and remark any element of a model.

# Structural Things

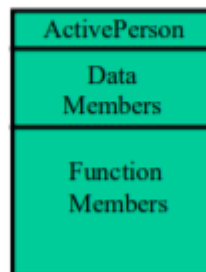
## Class

- A common description of a set of objects.



## Active Class

- A class whose objects can initiate a thread or a process.



## Interface

- A collection of operations provided by a class of a component.



## Use Case

- A sequence actions. A structure superimposed on top of behavioral things.



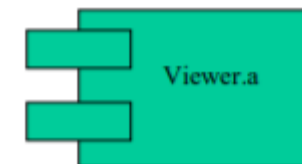
## Collaboration

- A collection of structural elements and behavioral elements.



## Component

- A physical packaging of classes, interfaces, and collaborations.



# Behavioral things

- Interactions: The “verbs” of the model. A set of messages exchanged among a set of objects.

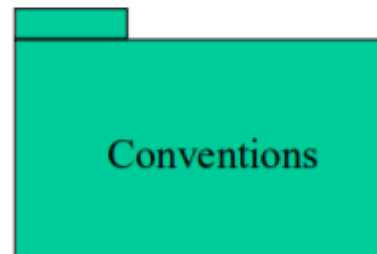


- State Machine: A sequence of states that an object goes through.



# Grouping things

- Package: a general-purpose mechanism for bundling together structural, behavioral, or other packages.



# Annotational things

- Notes: Explain the model, comments, constraints, etc.

This class  
is for ....

# Dependency

- A semantic relationship in which a change on one thing (the independent thing) may cause changes in the other thing (the dependent thing).

**Symbol**

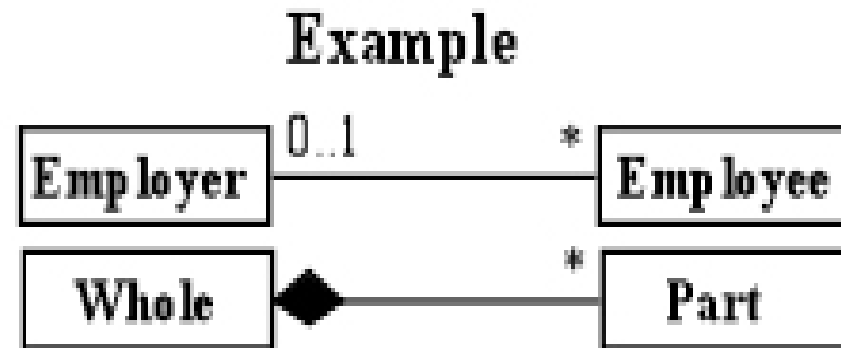
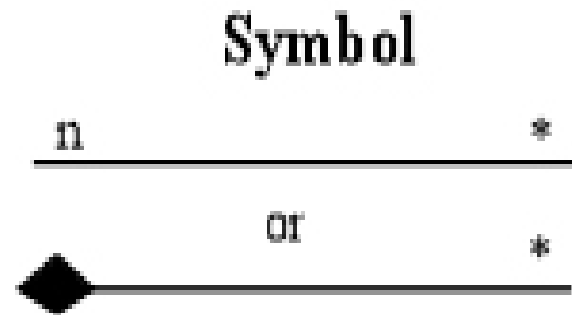


**Example**



# Association

- A structural relationship describing links between objects. May also include labels to indicate number and role of the links. In the example there may be any number of employees (\*) each of which has 0 or 1 employer. The double arrowhead is used to indicate a "has-a" relationship, meaning there is 1 employer who may have many (\*) employees.





# Generalization

A specialization/generalization relationship. Simply put this describes the relationship of a parent class (generalization) to its subclasses (specializations).

**Symbol**

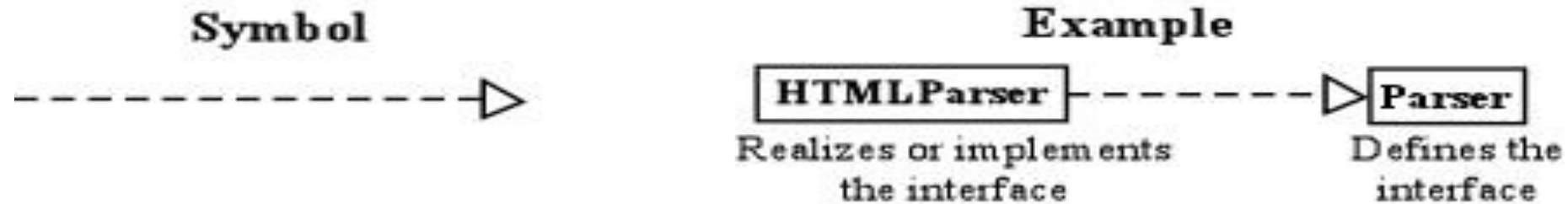


**Example**



# Realization

- Defines a relationship in which one class specifies something that another class will perform. Example: The relationship between an interface and the class that realizes or executes that interface.



# UML And Modeling

Module 2:UML diagrams overview:

# A UML diagram Overview

A **UML diagram** is a partial graphical representation (view) of a model of a system under design, implementation, or already in existence. UML diagram contains **graphical elements** (symbols) - UML nodes connected with edges (also known as paths or flows) - that represent elements in the UML model of the designed system. The UML model of the system might also contain other documentation such as use cases written as templated texts.

- The UML model of the designed system. The UML model of the system might also contain other documentation such as use cases written as templated texts.

# A UML diagram Overview

The **kind of the diagram** is defined by the primary graphical symbols shown on the diagram.

For example,

A diagram where the primary symbols in the contents area are classes is [class diagram](#).

A diagram which shows [use cases](#) and [actors](#) is [use case diagram](#).

A [sequence diagram](#) shows sequence of message exchanges between [lifelines](#)

# A UML diagram Overview

- UML specification does not preclude **mixing** of different kinds of diagrams
- To combine structural and behavioral elements to show a state machine nested inside a use case.
- Consequently, the boundaries between the various kinds of diagrams are not strictly enforced.
- At the same time, some **UML Tools** do restrict set of available graphical elements which could be used when working on specific type of diagram.

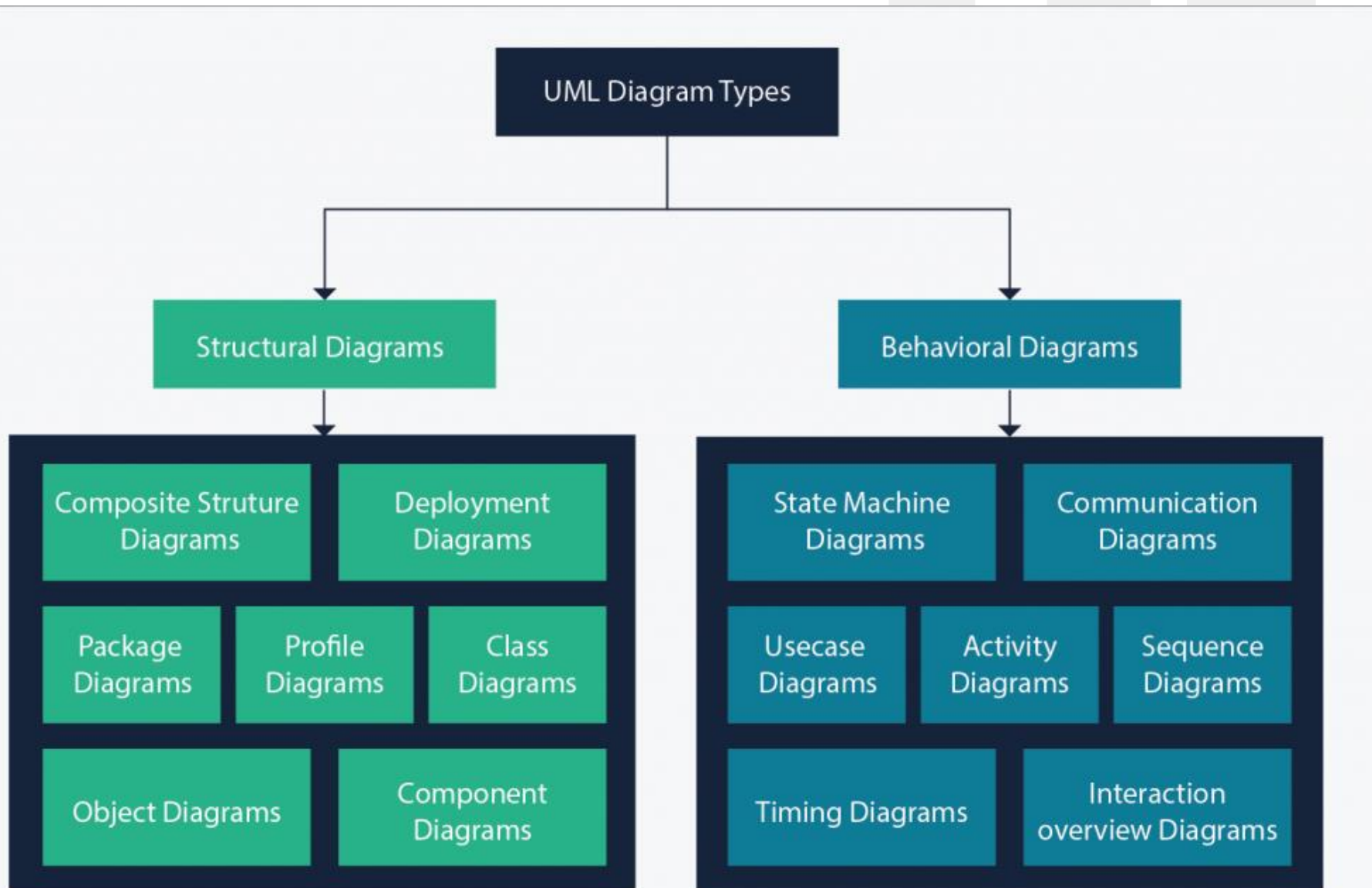
- UML includes nine diagrams - each capturing a different dimension of a software-system architecture.

# UML And Modelling

## Module 2:UML diagram types



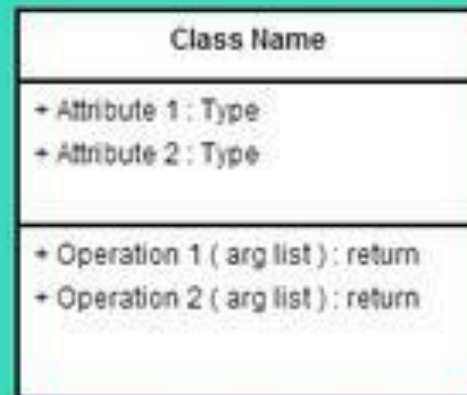
# Types of UML Diagram?



# UML DIAGRAM TYPES

## STRUCTURAL DIAGRAMS

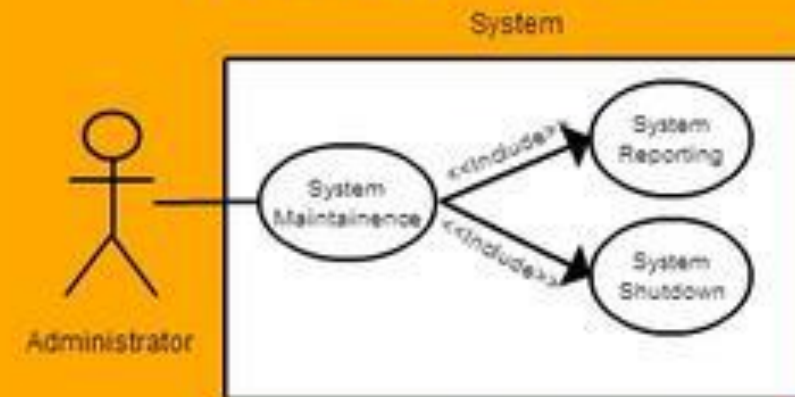
### CLASS DIAGRAM



Class shows the classes in a system, attributes and operations of each class and the relationship between each class.

## BEHAVIORAL DIAGRAMS

### USE CASE DIAGRAMS

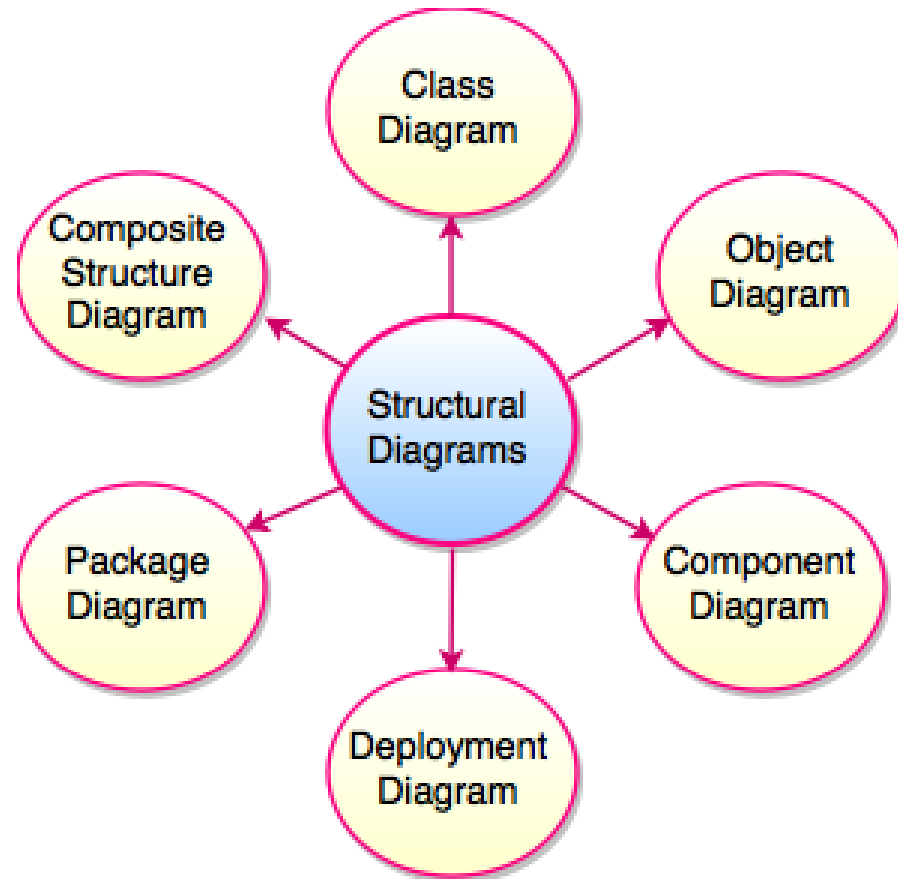


Use case diagrams gives a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions are interacted.

# Structural Diagram

- Structural diagram is an important part of the UML.
- It represents the static aspect of the system and the static parts of the diagrams are represented by classes, interfaces, objects, components and nodes.
- These diagrams show the things and different objects in a system being modeled.

# Types Of Structural Diagram



# Class Diagram

- Main building block of any object oriented solution.
- Shows the classes in a system, attributes, relationships and operations of each class.
- It represents the object orientation of a system. So it is used for development purpose.
- Most widely used diagram at the time of system construction.
- Used for data modeling.
- Shows the structure of the designed system at the level of classes and interfaces.

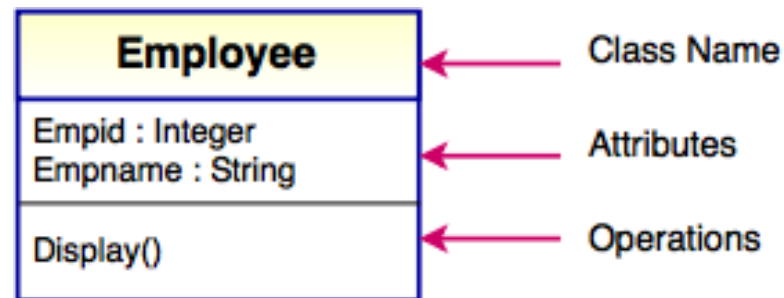
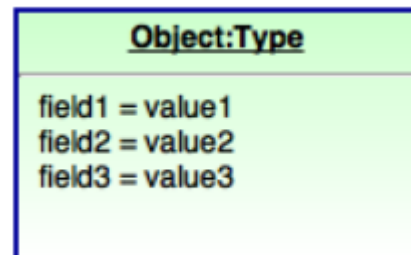


Fig. Class Diagram

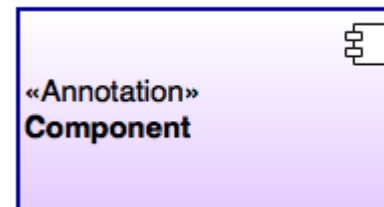
# Object Diagram

- Object diagram is derived from class diagram, so the object diagrams are dependent upon class diagrams.
- This diagram represents an instance of a class diagram.
- Object diagram is used to render a set of objects and their relationships as an instance.
- It shows a snapshot of the detailed state of a system at a point in time.
- It is used to show how a system will look like at a given time, because there is data available in the objects.



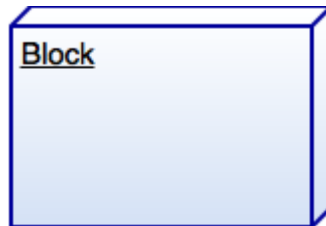
# Component Diagram

- Component diagram is used to illustrate the structure of arbitrarily complex systems.
- It is used in Component-Based Development (CBD) to describe systems with Service-Oriented Architecture (SOA).
- It shows components, provided and required interfaces, ports and relationships between them.
- Component diagram consists of classes, interfaces or collaborations.
- It is used to visualize the implementation.
- The components are communicate with each other using interfaces and the interfaces are linked using connectors.



# Deployment Diagram

- Deployment diagram is used to visualize the topology of the physical components of a system where the software components are deployed.
- It is used to describe the static deployment view of a system.
- These diagrams are used for describing the hardware components where software components are deployed.
- It helps to model the physical aspect of an object-oriented software system.
- It models the run-time configuration in a static view and visualizes the distribution of components in an application.
- It helps to model the physical aspect of an object-oriented software system.





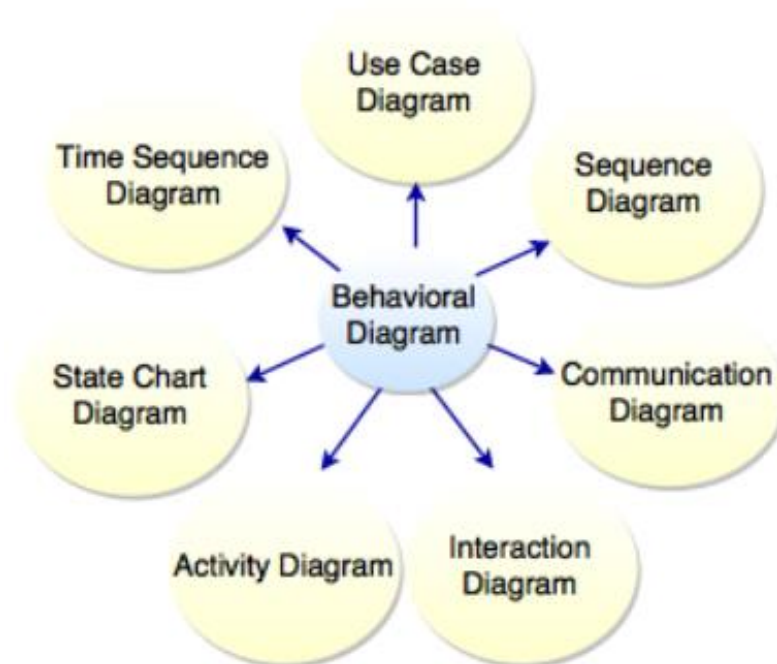
# Package Diagram

- Package diagram is a namespace used to group together elements that are semantically related and might change together.
- These diagram shows the arrangement and organization of model elements in middle to large scale project.
- It can show both structure and dependencies between sub-systems or modules.
- It represents different layers of a software system to illustrate the layered architecture of a software system and compile time grouping mechanism.
- It is used in large scale systems to picture dependencies between major elements in the system.



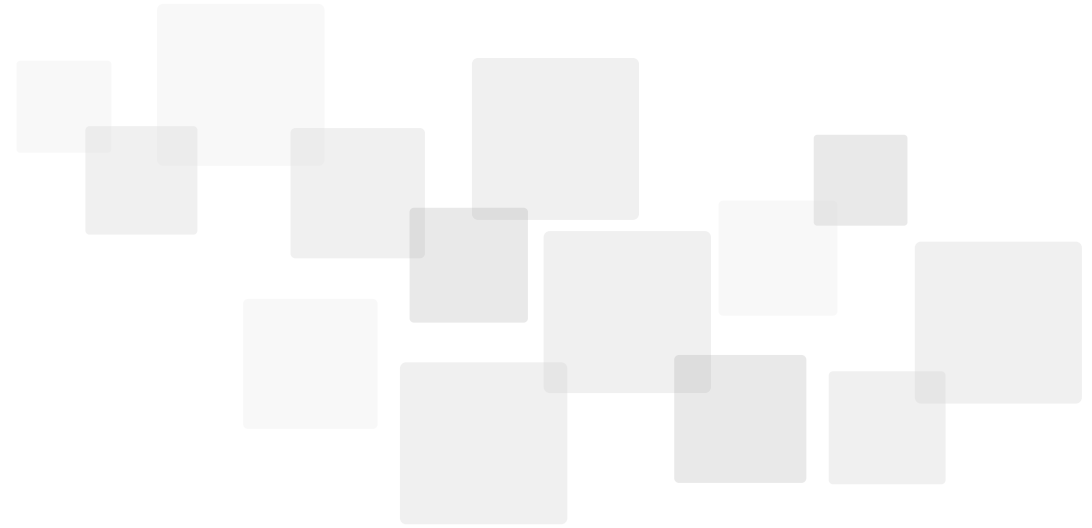
# Behavioural Diagrams

- Behavioral diagram is used to describe how the objects interact with each other to create a functioning system.
- These diagrams shows how would happen in a system.
- It is used to visualize, specify, construct and document the dynamic aspects of a system.



# Relationships

- Relationships help us connect the model elements.
- After finding out the primary Use Cases, one can start looking “into” the system to see if there are any relationships between the Use Cases.
- The following types of relationships can exist between the Use Cases:
  - include
  - Extend
- Will Be covered with more details after Use Case Diagram.



## Module 4: Activity, Use-case, Class and Sequence diagrams

# Use Case Diagram

- The notation for a use case diagram is pretty straightforward and doesn't involve as many types of symbols as other UML diagrams



Actor

**Actors:** Stick figures that represent the people actually employing the use cases.



Use Case

**Use cases:** Horizontally shaped ovals that represent the different uses that a user might have.



# Notations

## System



**System boundary boxes:** A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system. For example, Psycho Killer is outside the scope of occupations in the chainsaw example found below.

## Package Name

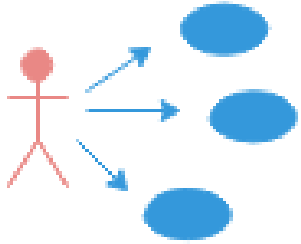


**Packages:** A UML shape that allows you to put different elements into groups. Just as with component diagrams, these groupings are represented as file folders.

**Associations:** A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.

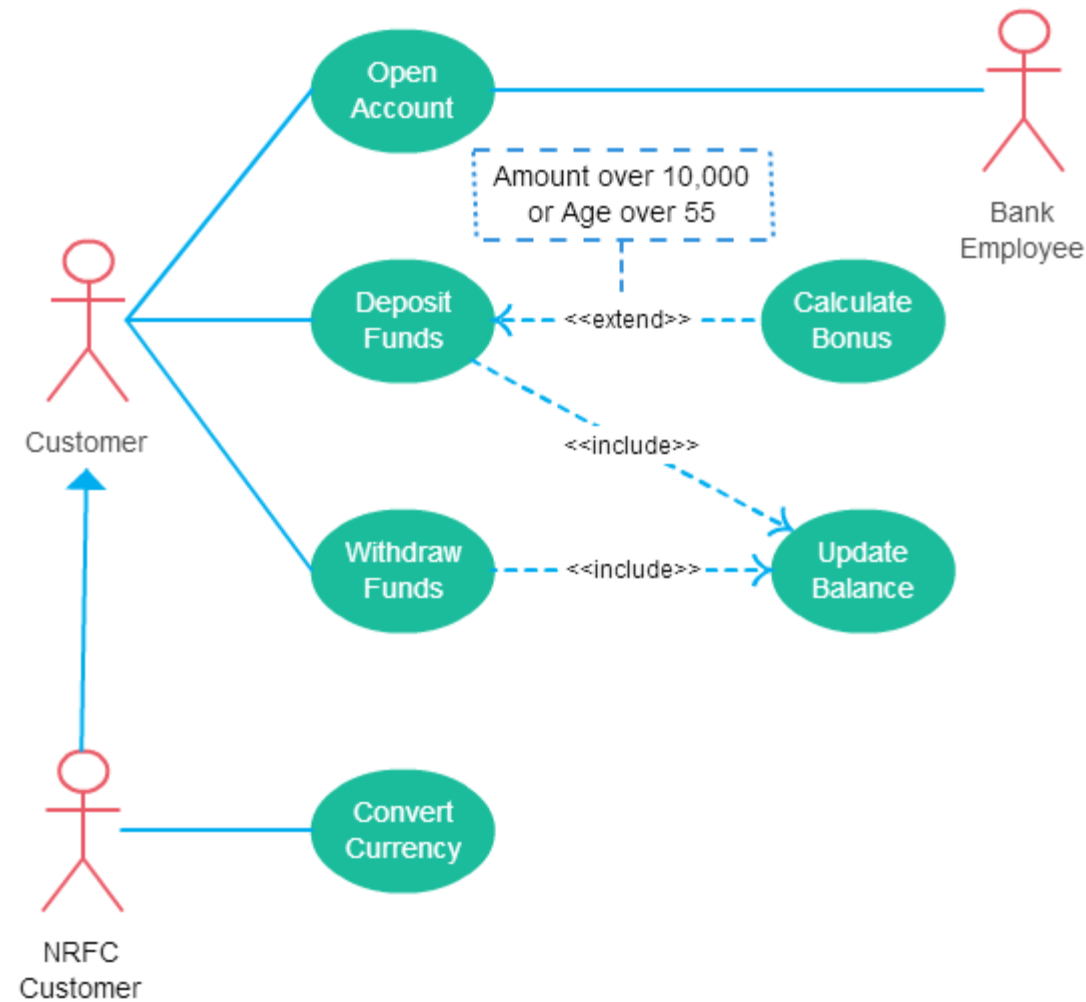


# Use case diagram



- They describe "WHAT" of a system rather than "HOW" the system does it.
- They are used to identify the primary elements and processes that form the system.
- The primary elements are termed as "actors" and the processes are called "use cases".
- Use Case diagrams shows "actors" and their "roles".

# Use Case Example





## Example : use case Diagram

- **Identifying Actors**

- Actors are **external entities** that interact with your system. It can be a **person**, another **system** or an **organization**.
  - In a banking system, the most obvious actor is the **customer**. Other actors can be **bank employee** or **cashier** depending on the role you're trying to show in the use case.

- **Identifying Use Cases**

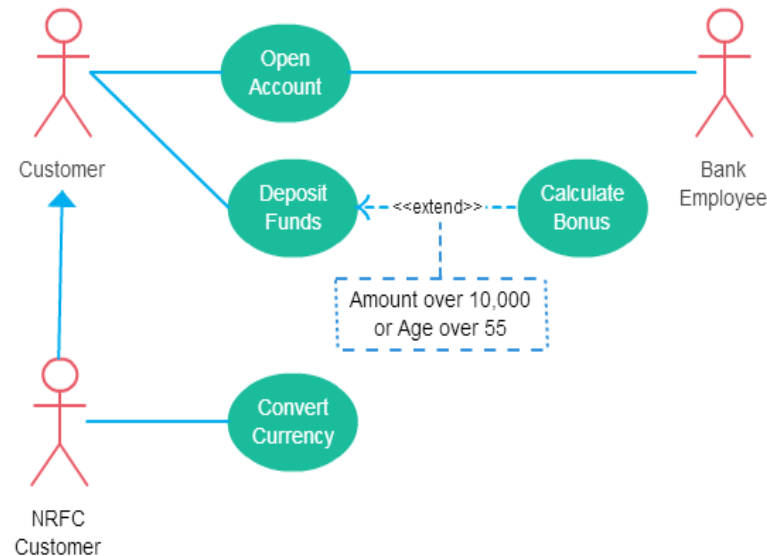
- A good way to do this is to identify what the actors need from the system.
  - In a banking system, a customer will need to **open accounts**, **deposit** and **withdraw funds**, **request check books** and similar functions.

## Example: use case Diagram

- **Identifying Include:** The invocation of a use case by another one.

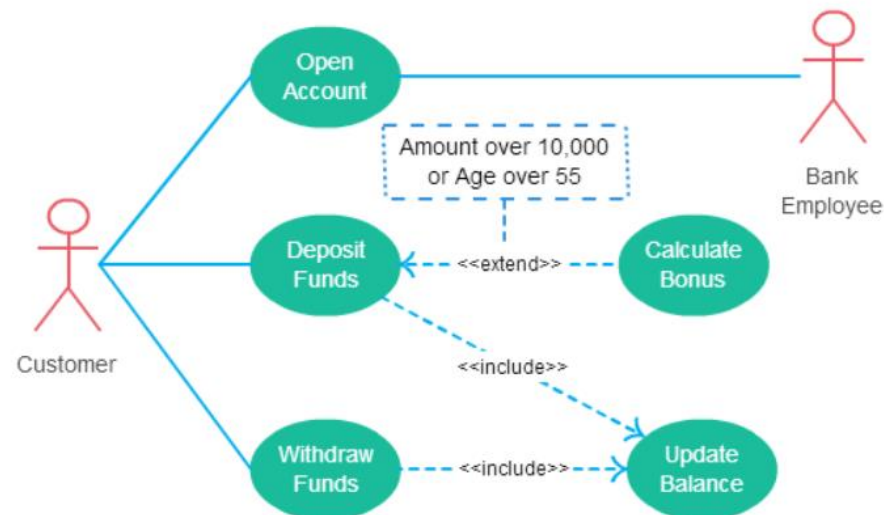
In an Include relationship, one Use Case includes behavior specified by another Use Case. If there are common steps in the scenarios of many Use Cases, they can be factored out into a separate Use Case. This Use Case can then be included as part of the “Primary Use Case”.

- Look for Common Functionality to use Include



**Identify Exclude** :inserting additional action sequences into the base use-case sequence.  
Here Use Case may be required by another use case based on “some condition”, or due to “some exceptional situation”.

There are some **functions that are triggered** optionally.  
In such cases, you can use the **extend relationship** and attach an extension rule to it.  
In the below banking system example “Calculate Bonus” is optional and only triggers when a certain condition is matched.

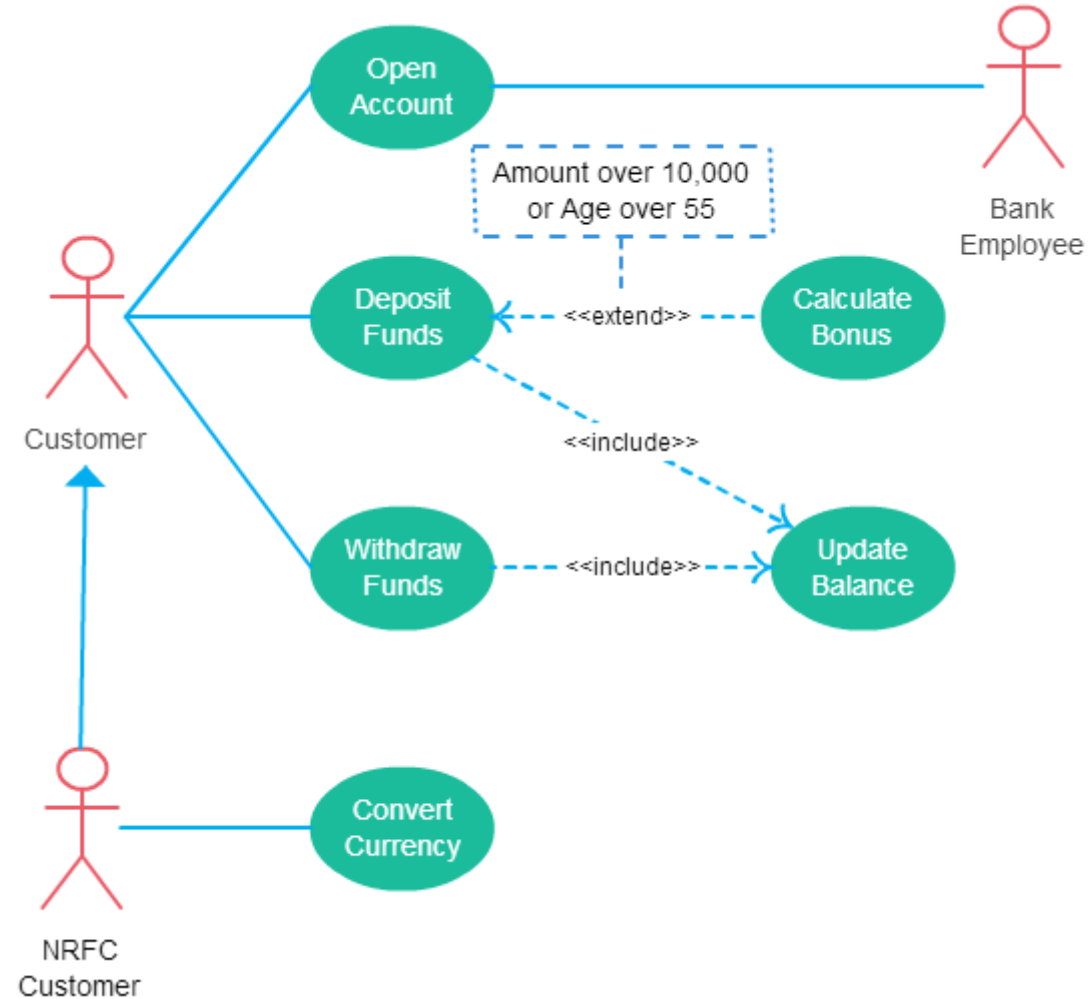


## • Optional Functions or Additional Functions

Is it Possible to **Generalize Actors** and **Use Cases**

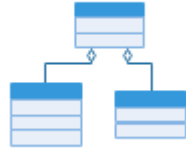
- There may be instances where actors are associated with similar use cases while triggering few use cases unique only to them. In such instances, you can generalize the actor to show the inheritance of functions. You can do a similar thing for use case as well.
- One of the best examples of this is “**Make Payment**” use case in a payment system.
- You can further **generalize** it to “Pay by Credit Card”, “Pay by Cash”, “Pay by Check” etc

## Example : Use case Diagram



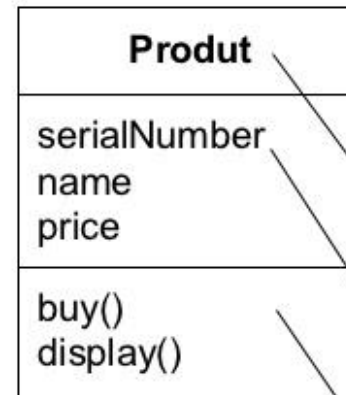
# Class Diagrams

- A class diagram shows a set of classes, interfaces, and collaborations and their relationships.
- These diagrams are the most common diagram found in modeling object-oriented systems.
- Class diagrams address the static design view of a system.
- Class diagrams that include active classes address the static process view of a system.
- Component diagrams are variants of class diagrams.



## Classes

A class is a template for actual, in-memory, instances



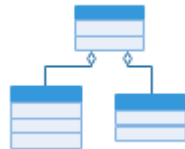
Class Name

Attributes

Operations

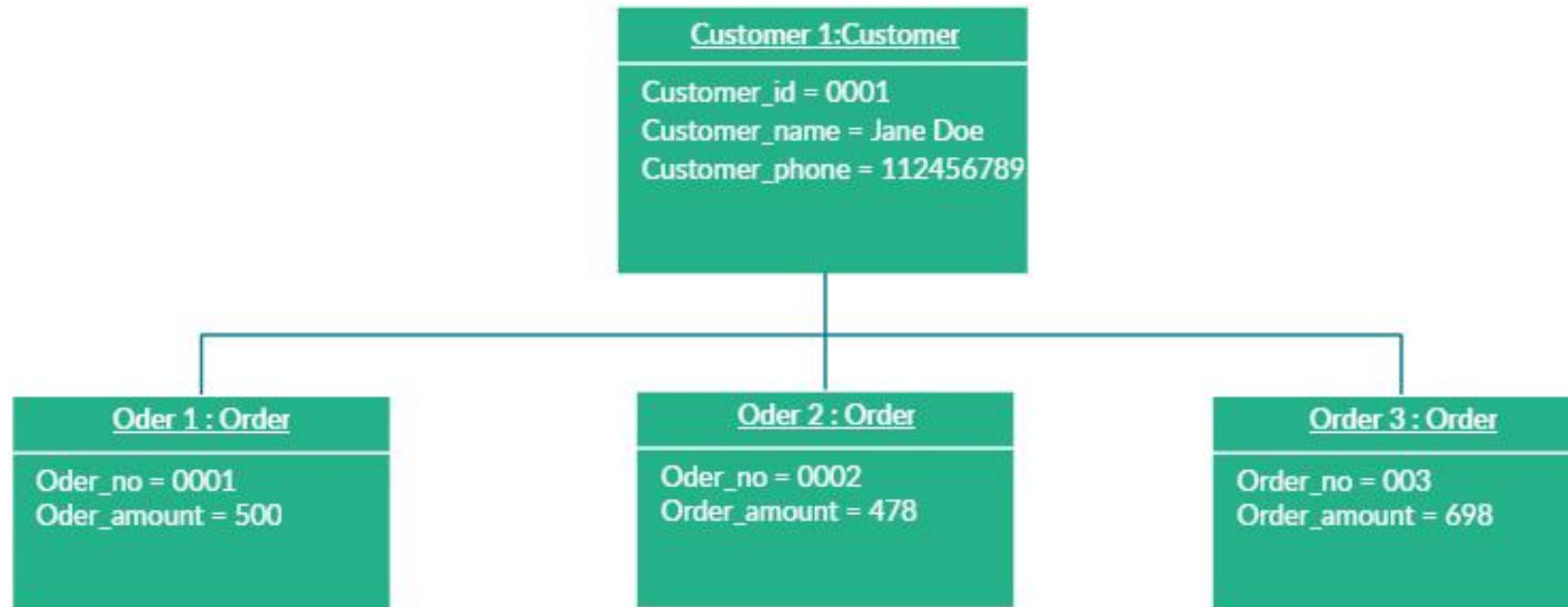
## Class diagram:

- From the use case diagram, we can now go to detail design of system, for which the primary step is class diagram.
- The best way to identify classes is to consider all "NOUNS" in use cases as classes, "VERBS" as methods of classes, relation between actors can then be used to define relation between classes.
- The relationship or association between the classes can be either an "is-a" or "has-a" relationship which can easily be identified from use cases.



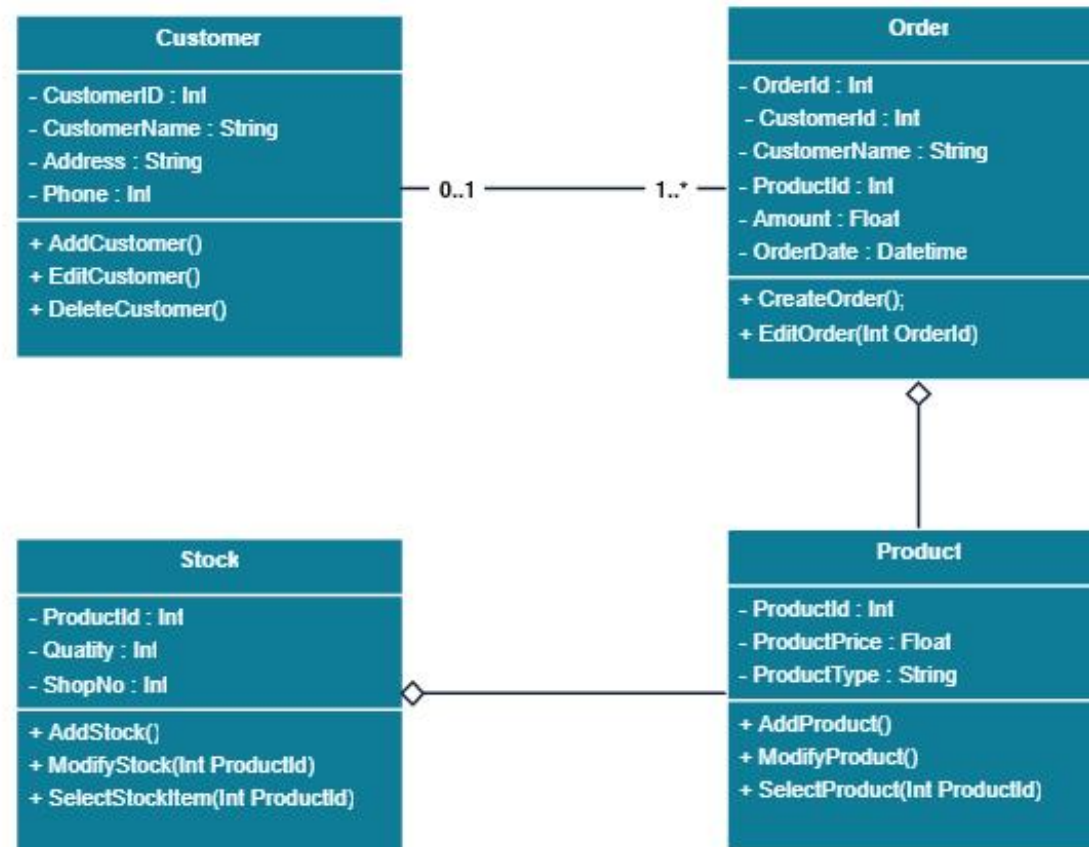


# Example Class Diagram



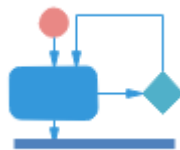
# Example Class Diagram

Class Diagram for Order Processing System

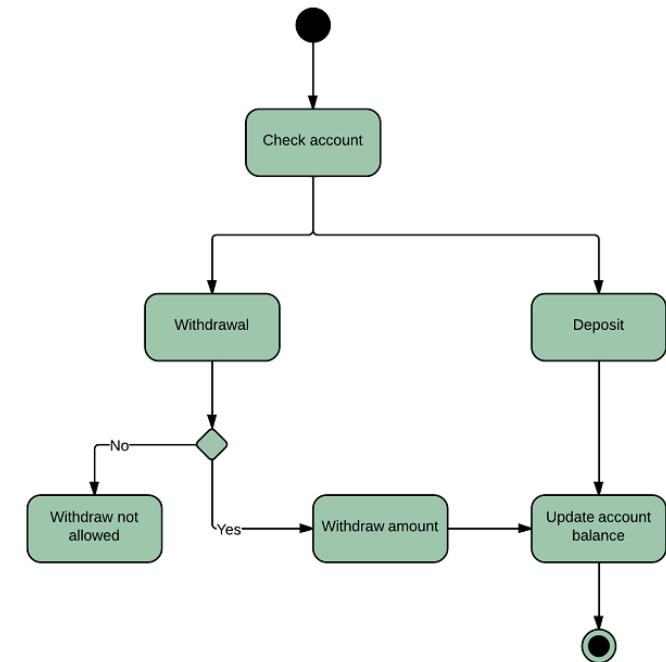
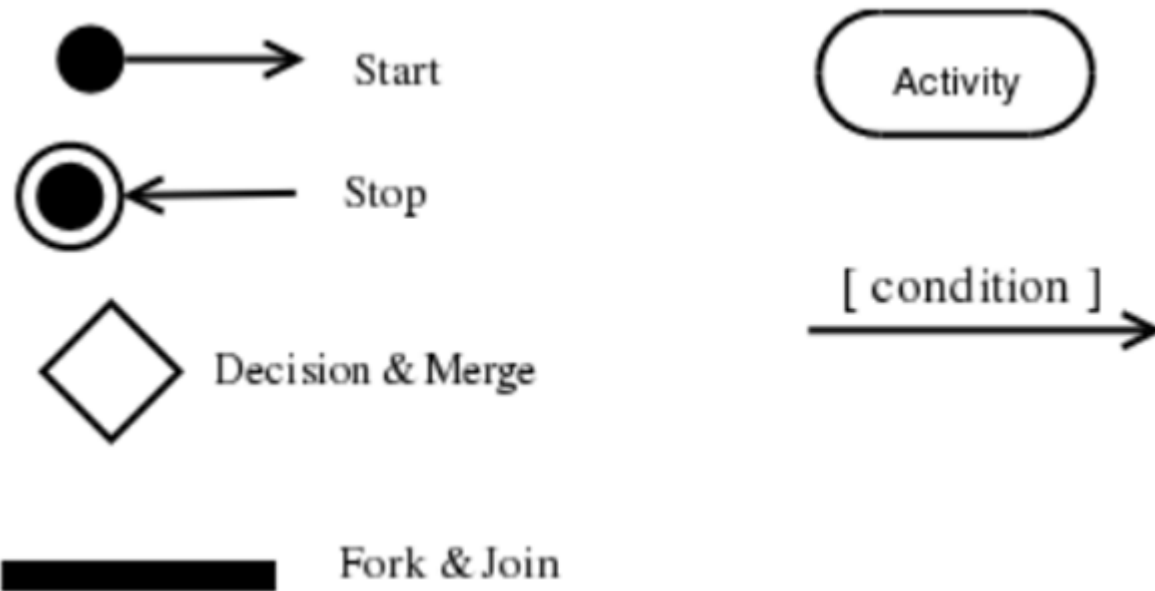


# Activity diagram

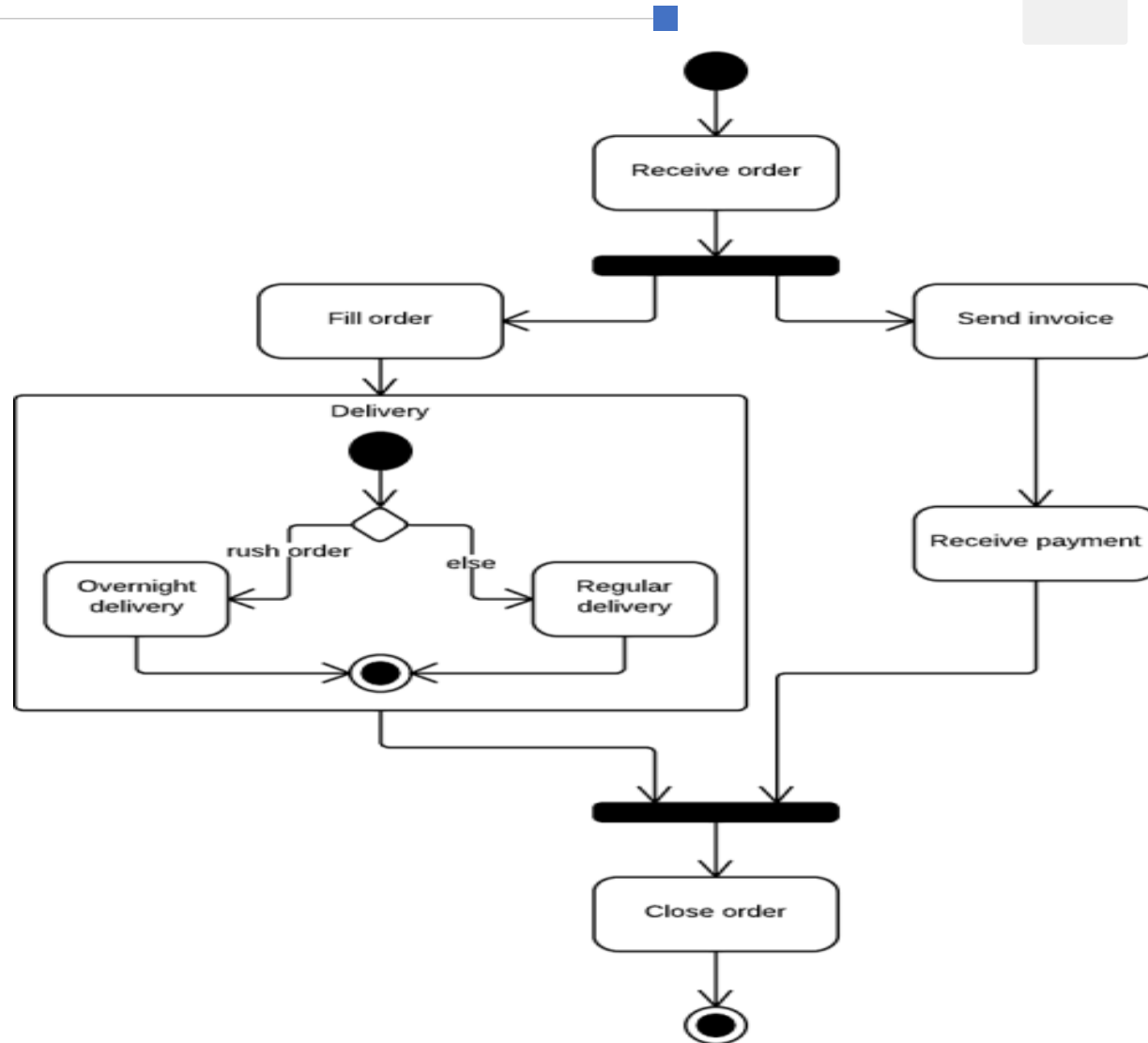
- Activity diagram is typically used for business process modeling, for modeling the logic captured by a single use case, or for visualizing the detailed logic of a business rule.
- Complicated process flows in the system are captured in the activity diagram.
- Activity diagrams give detail view of business logic.
- An activity diagram shows the structure of a process or other computation as the flow of control and data from step to step within the computation.
- Activity diagrams address the dynamic view of a system. They are especially important in modeling the function of a system and emphasize the flow of control among objects.



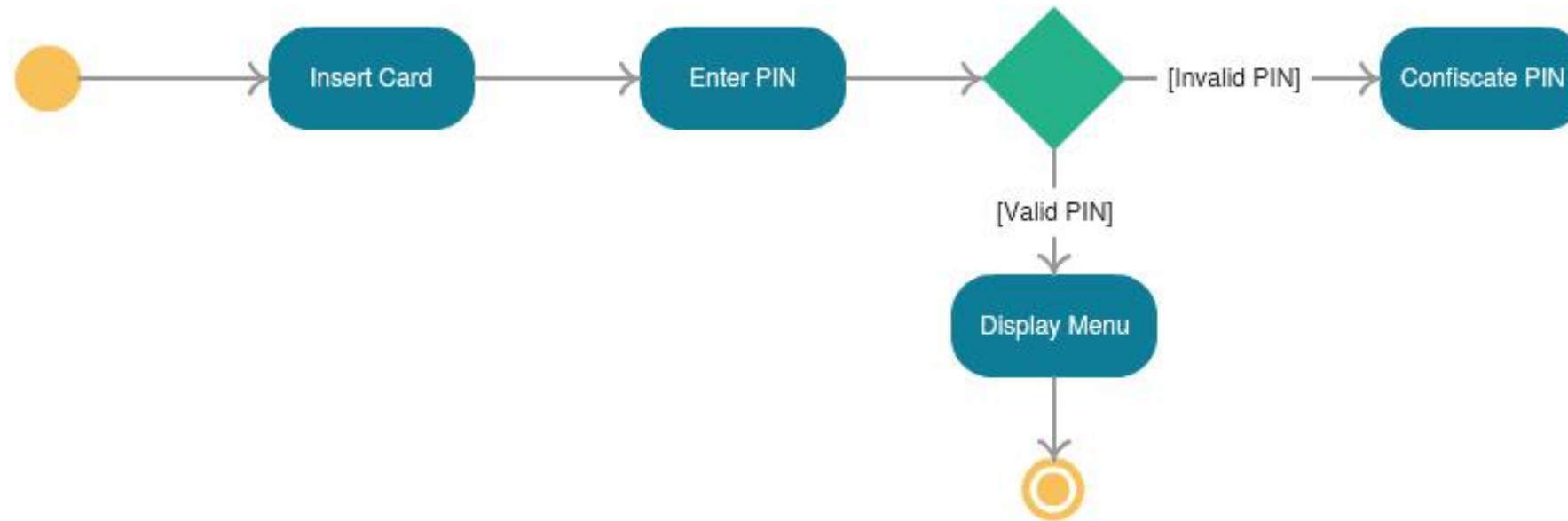
# Basic Notations used in Activity Diagram



# Activity Diagram

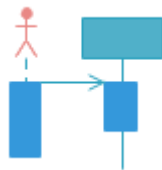


# Activity Diagram



# 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 different parts of a system work in a 'sequence' to get something done.



## Sequence diagram:

- They are called sequence diagrams because sequential nature is shown via ordering of messages.
- First message starts at the top and the last message ends at bottom.
- The important aspect of a sequence diagram is that it is time-ordered. This means that the exact sequence of the interactions between the objects is represented step by step. Different objects in the sequence diagram interact with each other by passing "messages".
- A sequence diagram is an interaction diagram that emphasizes the time-ordering of messages.



# Sequence Diagram Notations



Actor



Entity



Boundary



Control

# Sequence Diagram

- Before drawing the sequence diagram, it's necessary to identify the objects or actors that would be involved in creating a new user account. These would be;

- Librarian
- Online Library Management system
- User credentials database
- Email system

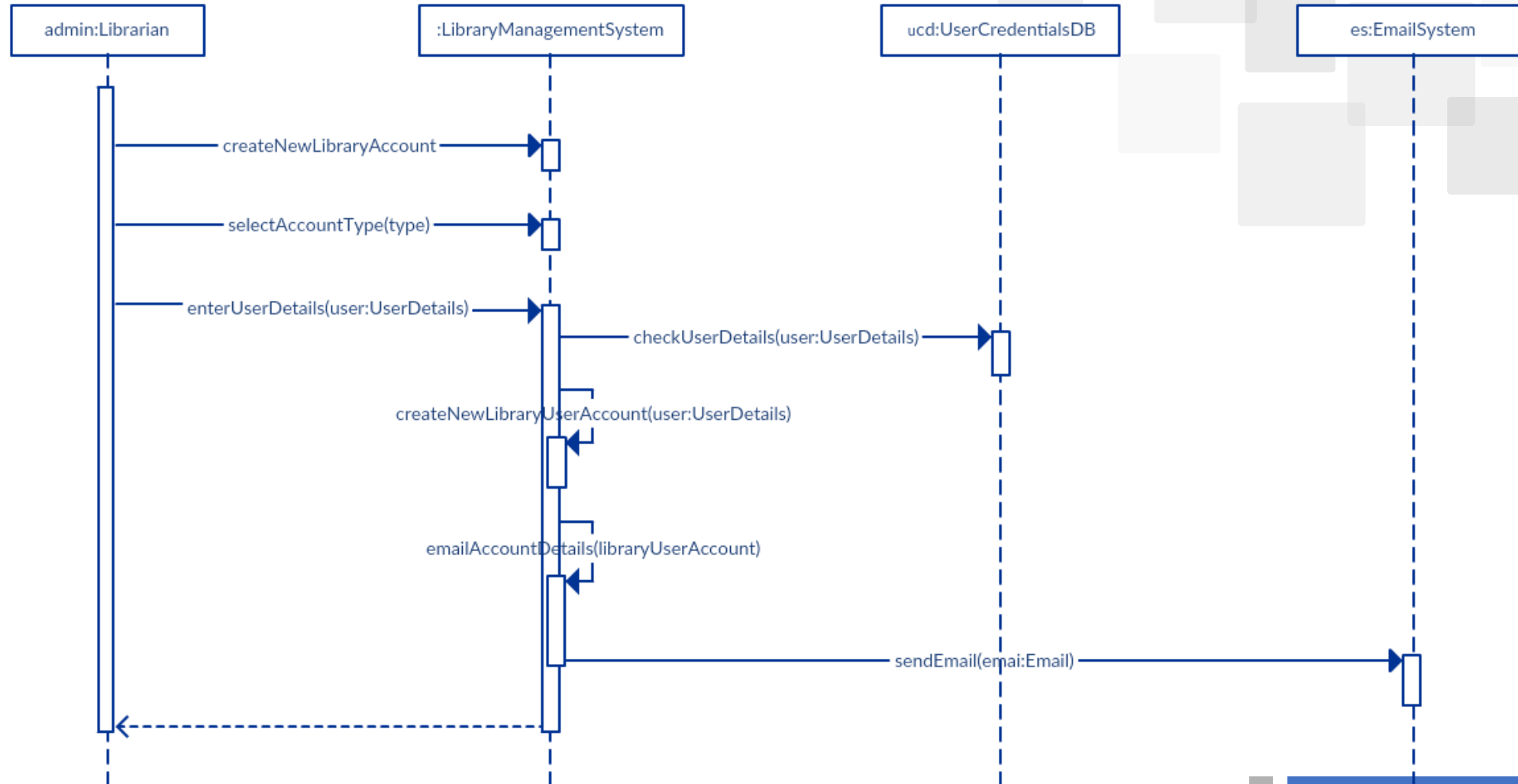
# Sequence Diagram

- Then write a detailed description on what the use case does.
- From this description, you can easily figure out the interactions that would occur between the objects above, once the use case is executed.

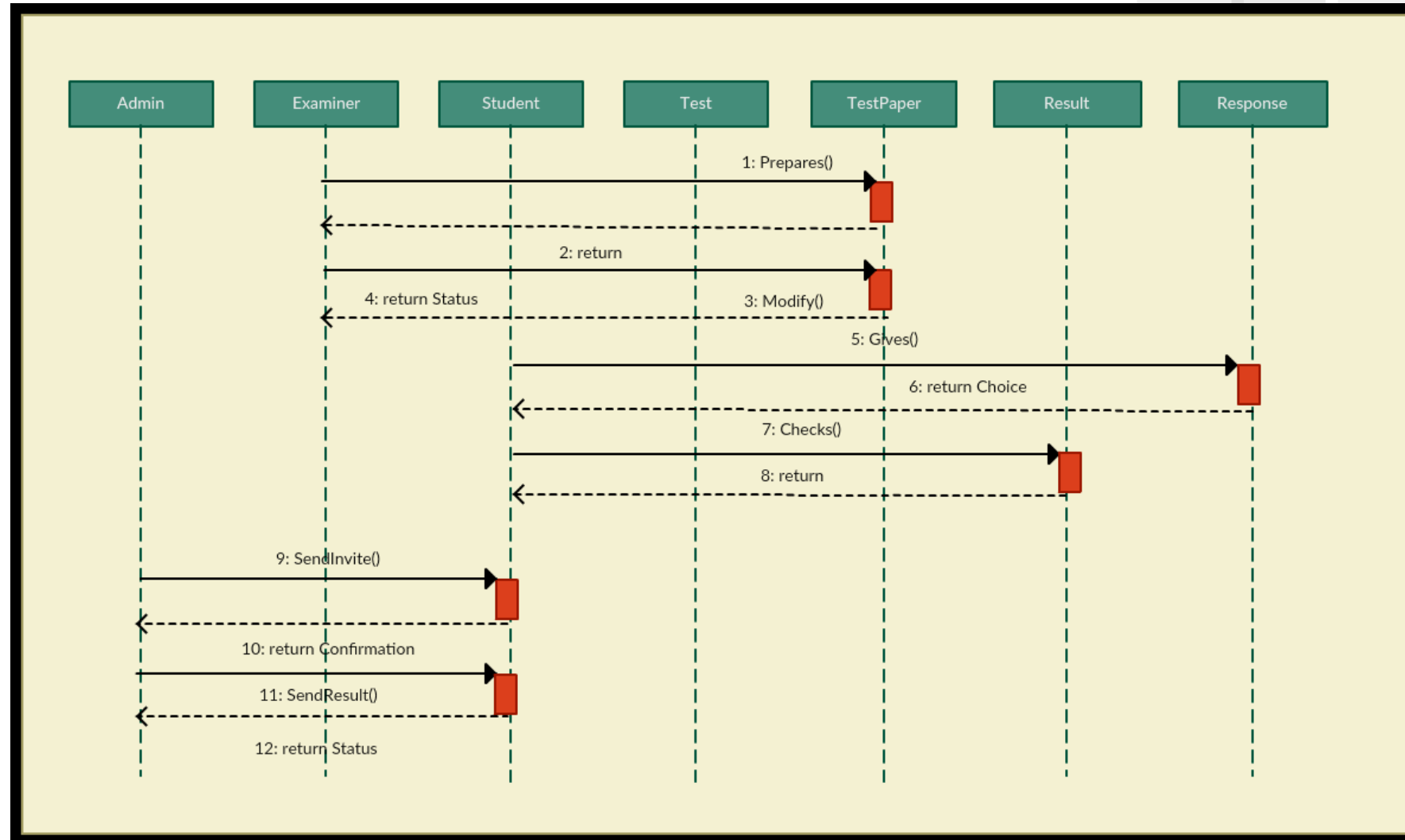
Here are the steps that occur in the use case named 'Create New Library User Account'.

- The librarian request the system to create a new online library account
- The librarian then selects the library user account type
- The librarian enters the user's details
- The user's details are checked using the user Credentials Database
- The new library user account is created
- A summary of the of the new account's details are then emailed to the user

# Sequence Diagram



# Sequence Diagram



Thank You

