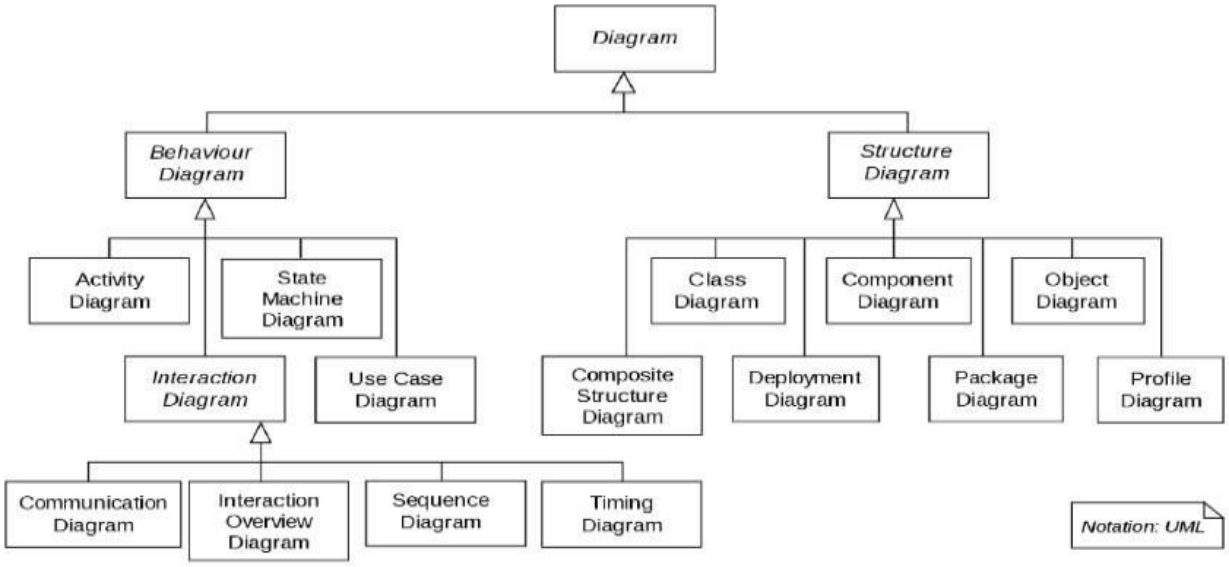
INTRODUCTION TO SOFTWARE ENGINEERING

UML stands for Unified Modeling Language. It's a rich language to model software solutions, application structures, system behavior and business processes. There are 14 UML diagram types to help model these behaviors.



STRUCTURAL DIAGRAMS

Class diagram:

A UML class diagram is a fundamental building block of any object-oriented solution. It depicts a static, object-oriented system, defining projects by classes, attributes, and functions. In other words, it shows the classes within a system and the operations of each one. Both software engineers and business managers use this interaction diagram to model different connections involved within a process.

In the diagram, the class is represented by a rectangle. Each rectangle is split vertically into three sections. The top section has the name of the class, while the second and third sections provide details about class operations, behaviors, and attributes.

Package diagram:

As the name suggests, a package diagram shows the dependencies and relationships between packages in a system. A package refers to anything that organizes any model element. In the diagram itself, it's presented as a file folder. The package diagram allows the user to group items into folders.

Items that can be grouped include classes, use cases, documents, diagrams, and even other packages. Once each item is allocated to a package, all packages are arranged and ranked hierarchically in the diagram. Because they easily portray the top-level structure, package diagrams are mostly useful for visualizing software systems.

Object diagram:

Object diagrams show the attributes of different objects within a system. And like class diagrams, they show the relationships between the objects in a piece of software. The difference is that object

diagrams use real-world examples. Object diagrams are also called instance diagrams because they display what the system looks like at a specific moment in time.

Component diagram:

A UML component diagram breaks down a complex system into smaller components, making it easier to visualize. Unlike other UML diagrams, it specifically outlines the relationship between these components. UML component diagrams give developers an overarching understanding of a system's physical objects. This type of UML diagram shows the structural relationship between each physical component and subcomponent in a complex software system. This helps stakeholders understand how the components are organized and wired together.

Deployment diagram:

A deployment diagram shows the configuration of processing nodes and all the components that live on them. It depicts the physical arrangement of the nodes in a distributed system. These diagrams are especially useful when you're building software that will operate on different hardware systems. System engineers are most likely to use these diagrams as they get to keep track of their entire hardware mesh and prepare the system for launch without any issues. It also allows them to see performance, maintainability, and scalability.

Composite structure diagram:

Composite structure diagrams visualize a class's internal structure. These diagrams break down the network of classes, interfaces, and components. This type of UML diagram also shows how these elements interact with each other and why they are essential to the overarching software structure.

Behavioral diagrams

Activity diagram:

A UML activity diagram is a flowchart that outlines all a system's activities. It shows everything from start to finish, defining the various decision paths and steps that need to happen to move from one activity to the next. The steps can be chronological, branched, or simultaneous.

You can use a UML activity diagram to home in on any one component and to get a high-level overview of the dynamic aspects of a system. Using an intuitive UML activity diagram maker is the best way to create these diagrams.

Sequence diagram:

UML sequence diagrams — sometimes known as event diagrams — show the order in which your objects interact. This includes the lifelines of your objects, the processes that interact with your objects, and the messages exchanged between the objects to perform a function.

Developers and business professionals often use these diagrams to understand how to structure a new system or improve an existing process. You can easily create them using an online UML sequence diagram.

Communication diagram:

UML communication diagrams are also called collaboration diagrams. They are relatively like sequence diagrams, focusing on the messages passed between different objects. While sequence diagrams look at processing over time, a communication diagram creates a complete, big-picture map of your product. A communication diagram tool makes it easy to create these visualizations.

Interaction overview diagram:

Like activity diagrams, interaction overview diagrams visualize the flow of activity and the sequence of those activities. The difference is that each activity in an interaction overview diagram is shown as a frame. The nodes in an interaction overview diagram represent the interactions in a system.

Timing diagram:

Timing diagrams are useful for deployments, as they depict the behaviour of specific objects within an explicit time frame. These diagrams are usually straightforward, but when dealing with more than one object, they show the interactions between many different and important sequences within that time frame.

Use case diagram:

Use case diagrams provide a graphic overview of the actors involved in a software system. By illustrating system functionality and outlining a system's expected behaviour, they help developers. analyze the relationships between use cases and personas. You can easily visualize these interactions using a use case diagram tool.

State machine diagram:

Also known as state chart diagrams, UML state machine diagrams show the behaviors of different components in a system. You can visualize how elements act differently according to the state of the program being developed. Create a state chart easily using Miro's state diagram tool.

ADVANTAGES OF UML

Readable and Flexible:

The best benefit is that the diagram's codes are easily readable to any programmer that understands even a tiny fraction of the program. With the help of this, a computer programmer can run the program with the codes' help and change or reuse them anytime he wants. UML is a much-needed program for software development because of how flexible it is to customize it according to the use of the company or technology.

Proper Communication for the Software Architecture:

The system's blueprint is the software architecture because it's the framework on which the process and the efficiency depend. UML (Unified Modelling Language) is the extensive language that is used for modelling software engineering. It also helps in assessing the performance of the users and with tracking, security, and gives relevant guidelines for the assigned operation. As it has properties to reach widely, UML is the perfect language to communicate visual information about the software architecture to many workers who use it.

Planning Tool before Making a Program:

UML can help you plan your program before it is time to start programming it. The tool can help with generating codes that can set up the model UML. The diagram is relatively easy to change and can help you reduce overhead when you are finally implementing the program.

Easily Debug Any Issue:

A more extensive program can take up to hours and hours of searching to find the error in it, and it can also cause problems in the system later. So, a program is designed nicely with the help of UML so that each of the tasks carries its own codes, which are easier for the programmer to debug.

DISADVANTAGES OF UML

It Is Just a Language:

You can say that it is just a language to communicate and the people who understand it can talk to each other. And it totally depends on how you are using it. If you are using it for a complete modelbased system, then only you can get information with regards to traceability, which is quite important in some industries, but it requires serious discipline. However, UML also lacks formal semantics in various areas. State machines and composite structures are two areas in which there is work that is being done to improve the quality of it. If you don't get its full advantage, then it can restrict you from accomplishing your goals that have been set by your company.

Designing – That Is All

UML consists of too much design that takes so much time to make and can get overwhelming for some companies. This results in companies to overlook the actual program and overanalyze the real issue by falling for the codes.

It Can Ascend Its Complexity

Some programmers think they are better off without it because it can grow in such complexity, making the new programmers stay out of learning or using it. This makes using the UML less. important because of its massive size of diagrams and codes.

EXPERIMENT 1

AIM: Create a user-centric use case diagram for the university site AMIZONE

TOOL USED: draw.io.

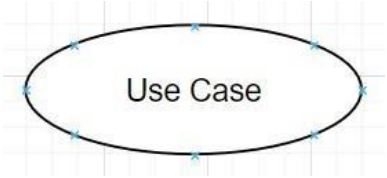
THEORY:

Use Case Diagram

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well.

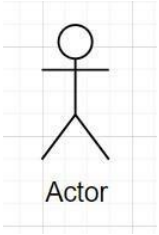
- **Use Case Diagrams and symbol Notations:**

  The use cases are represented by either circles or ellipses. Use cases represent the functionality of the system, as well as the end-goal of the actor. Use cases should be placed inside the system.



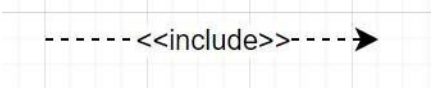
- **Actors:**

  Actors are external entities that interact with the system. The actors are often shown as stick figures. These can include users, other systems, or hardware devices. In the context of a Use Case Diagram, actors initiate use cases and receive the outcomes. Proper identification and understanding of actors are crucial for accurately modeling system behavior.



- **Include Relationship**

  The Include Relationship indicates that a use case includes the functionality of another use case. It is denoted by a dashed arrow pointing from the including use case to the included use case. This relationship promotes modular and reusable design.

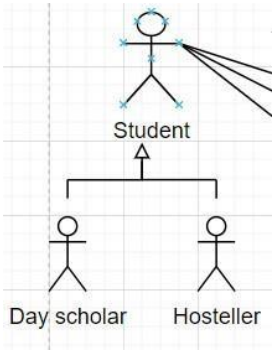


- **Extend Relationship**

  The Extend Relationship illustrates that a use case can be extended by another use case under specific conditions. It is represented by a dashed arrow with the keyword “extend.” This relationship is useful for handling optional or exceptional behavior.

- **Generalization Relationship**

  The Generalization Relationship establishes an "is-a" connection between two use cases, indicating that one use case is a specialized version of another. It is represented by an arrow pointing from the specialized use case to the general use case.



- **Association Relationship:**

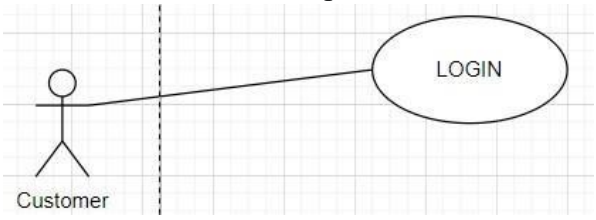
  The Association Relationship represents a communication or interaction between an actor and a use case. It is depicted by a line connecting the actor to the use case. This relationship signifies that the actor is involved in the functionality described by the use case.

  Example: Online Banking System

  Actor: Customer

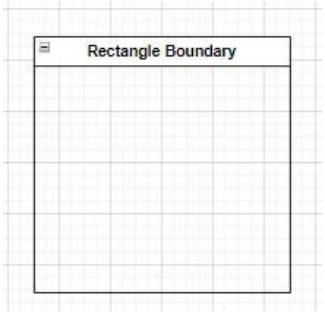
  Use Case: Transfer Funds

  Association: A line connecting the "Customer" actor to the "Transfer Funds" use case, indicating the customer's involvement in the funds transfer process.

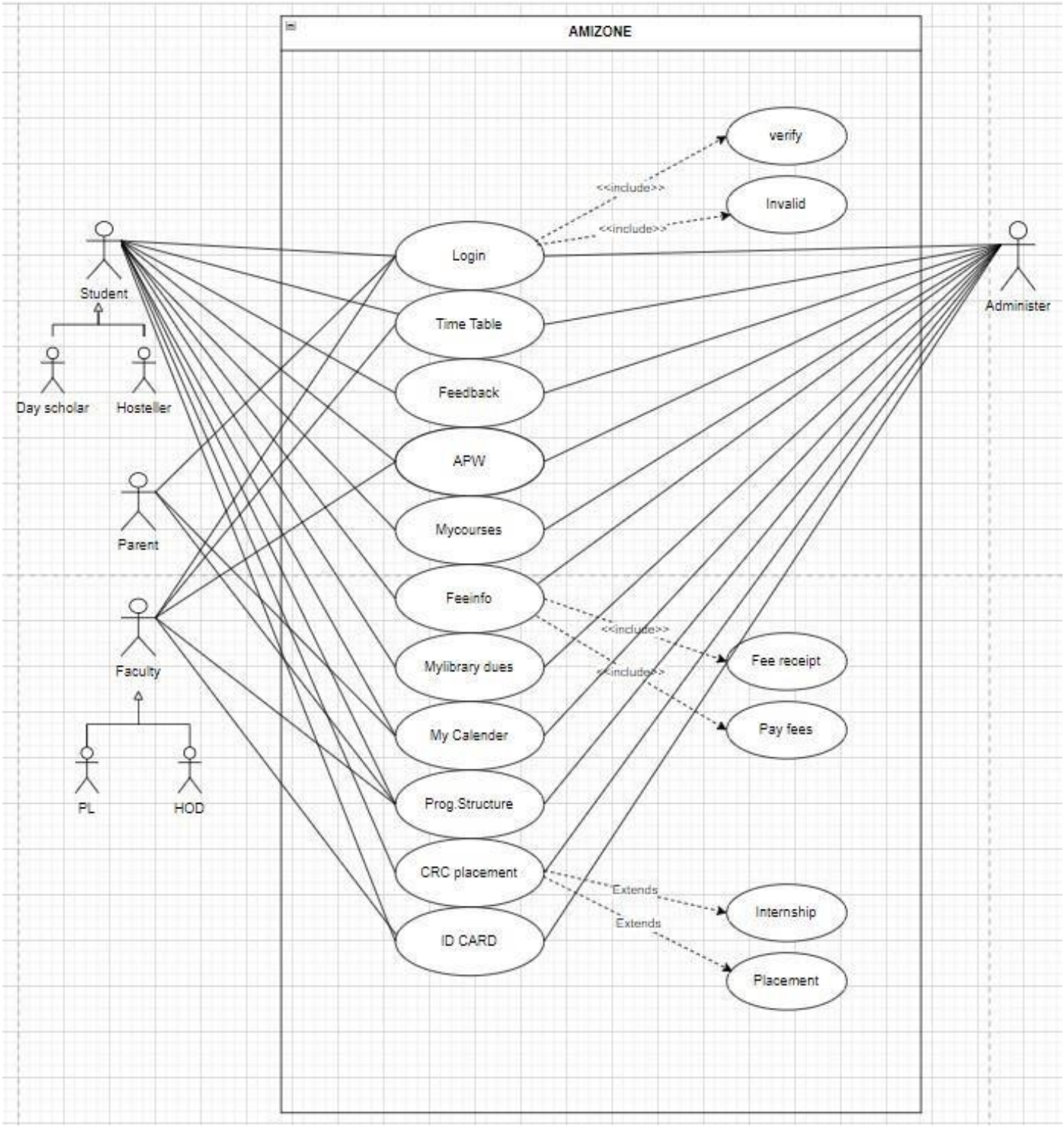


- **System Boundary:**

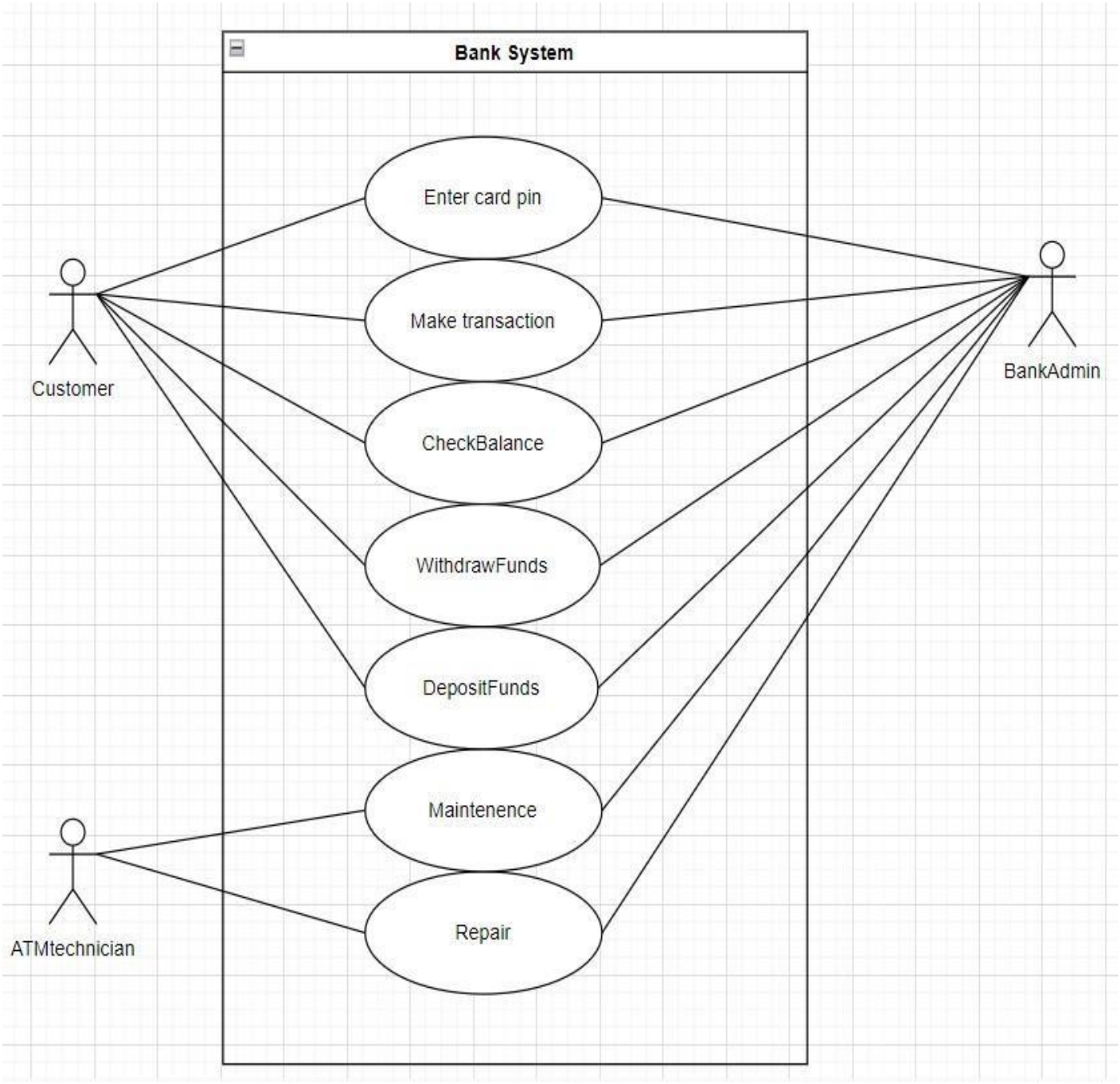
The system boundary is a visual representation of the scope or limits of the system you are modeling. It defines what is inside the system and what is outside. The boundary helps to establish a clear distinction between the elements that are part of the system and those that are external to it. The system boundary is typically represented by a rectangular box that surrounds all the use cases of the system.



USE CASE DIAGRAM FOR AMIZONE



USE CASE DIAGRAM FOR ATM SYSTEM


Bank System
Enter card pin
Make transaction
CheckBalance
WithdrawFunds
DepositFunds
Maintenence
Repair
Customer
BankAdmin
ATMtechnician

EXPERIMENT – 2

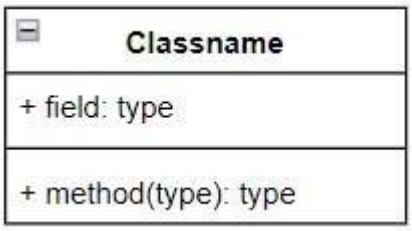
AIM: Draw a class diagram of ATM Machine.

THEORY:

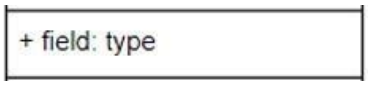
The class diagram describes the attributes and operations of a class and the constraints imposed on the system. The class diagrams are widely used in the modelling of object-oriented systems because they are the only UML diagrams which can be mapped directly with object-oriented languages. The class diagram shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a static diagram.

Class Diagram Annotations

- **Class:** The UML representation of a class is a rectangle containing three compartments stacked vertically. The top compartment shows the class's name. The middle compartment lists the class's attributes. The bottom compartment lists the class's operations.

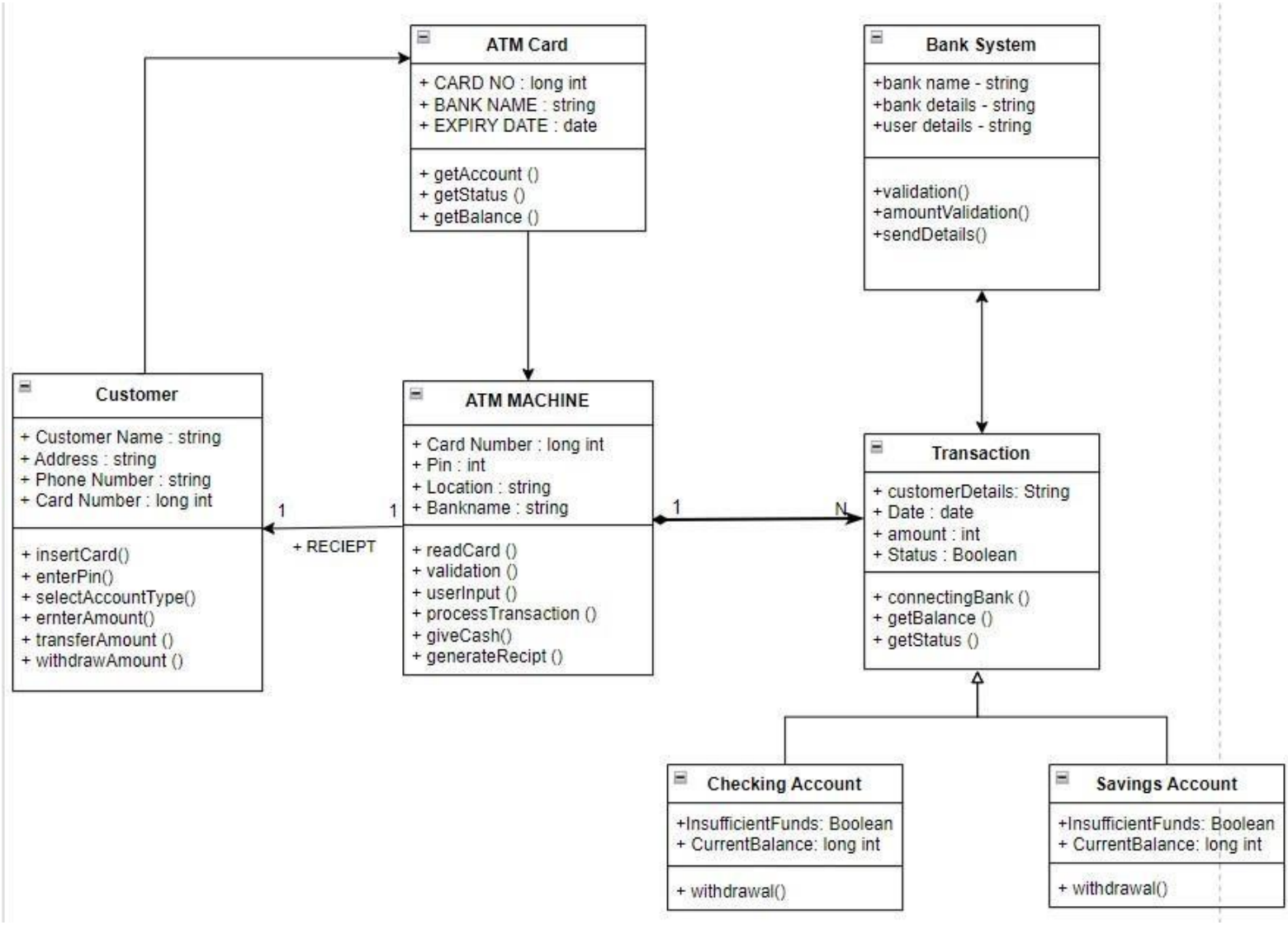


- **Class Attribute List:** The attribute section of a class (the middle compartment) lists each of the class's attributes on a separate line. The attribute section is optional, but when used it contains each attribute of the class displayed in a list format. The line uses the following format:



- **Inheritance:** A very important concept in object-oriented design, inheritance, refers to the ability of one class (child class) to inherit the identical functionality of another class (super class), and then add new functionality of its own. To model inheritance on a class diagram, a solid line is drawn from the child class (the class inheriting the behavior) with a closed, unfilled arrowhead (or triangle) pointing to the super class.

DIAGRAM:


ATM Card
+ CARD NO : long int
+ BANK NAME : string
+ EXPIRY DATE : date
+ getAccount ()
+ getStatus ()
+ getBalance ()
Bank System
+bank name - string
+bank details - string
+user details - string
+validation()
+amountValidation()
+sendDetails()
Customer
+ Customer Name : string
+ Address : string
+ Phone Number : string
+ Card Number : long int
+ insertCard()
+ enterPin()
+ selectAccountType()
+ ernterAmount()
+ transferAmount ()
+ withdrawAmount ()
ATM MACHINE
+ Card Number : long int
+ Pin : int
+ Location : string
+ Bankname : string
+ readCard ()
+ validation ()
+ userInput ()
+ processTransaction ()
+ giveCash()
+ generateRecipt ()
Transaction
+ customerDetails: String
+ Date : date
+ amount : int
+ Status : Boolean
+ connectingBank ()
+ getBalance ()
+ getStatus ()
1
1
+ RECIEPT
1
N
Checking Account
+InsufficientFunds: Boolean
+ CurrentBalance: long int
+ withdrawal()
Savings Account
+InsufficientFunds: Boolean
+ CurrentBalance: long int
+ withdrawal()

EXPERIMENT 3

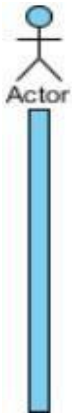
AIM: Draw Sequence Diagram of ATM

THEORY:

UML Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent the time what messages are sent and when.

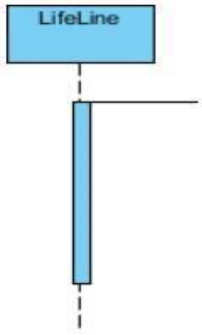
Sequence Diagram Annotations:

- **Actor:** A type of role played by an entity that interacts with the subject (e.g., by exchanging signals and data). Represents the roles played by human users, external hardware, or other subjects.



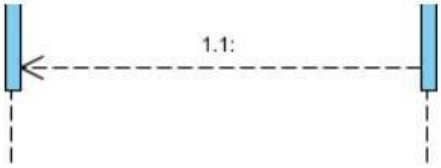
- **Lifeline:** A lifeline represents an individual participant in the Interaction.

- **Activations:** A thin rectangle on a lifeline) represents the period during which an element is performing an operation. The top and the bottom of the rectangle are aligned with the initiation and the completion time respectively.

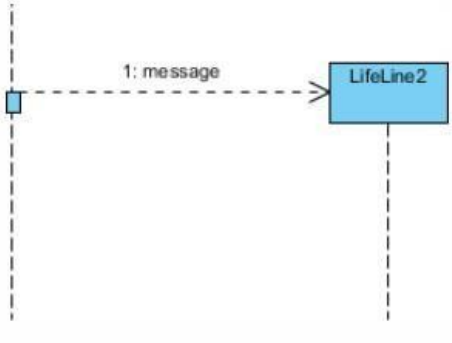


- **Call Message:** A message defines a particular communication between Lifelines of an Interaction. Call message is a kind of message that represents an invocation of operation of target lifeline.

- **Return Message:** A message defines a particular communication between Lifelines of an Interaction. Return message is a kind of message that represents the pass of information back to the caller of a corresponded former message.



- **Create Message:** A message defines a particular communication between Lifelines of an Interaction. Create message is a kind of message that represents the instantiation of (target) lifeline.



- **Destroy Message:** A message defines a particular communication between Lifelines of an Interaction. Destroy message is a kind of message that represents the request of destroying the lifecycle of target lifeline.

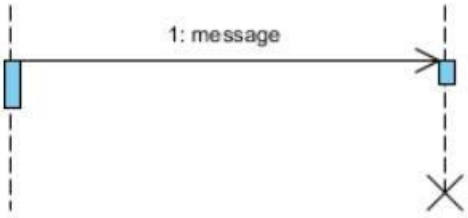
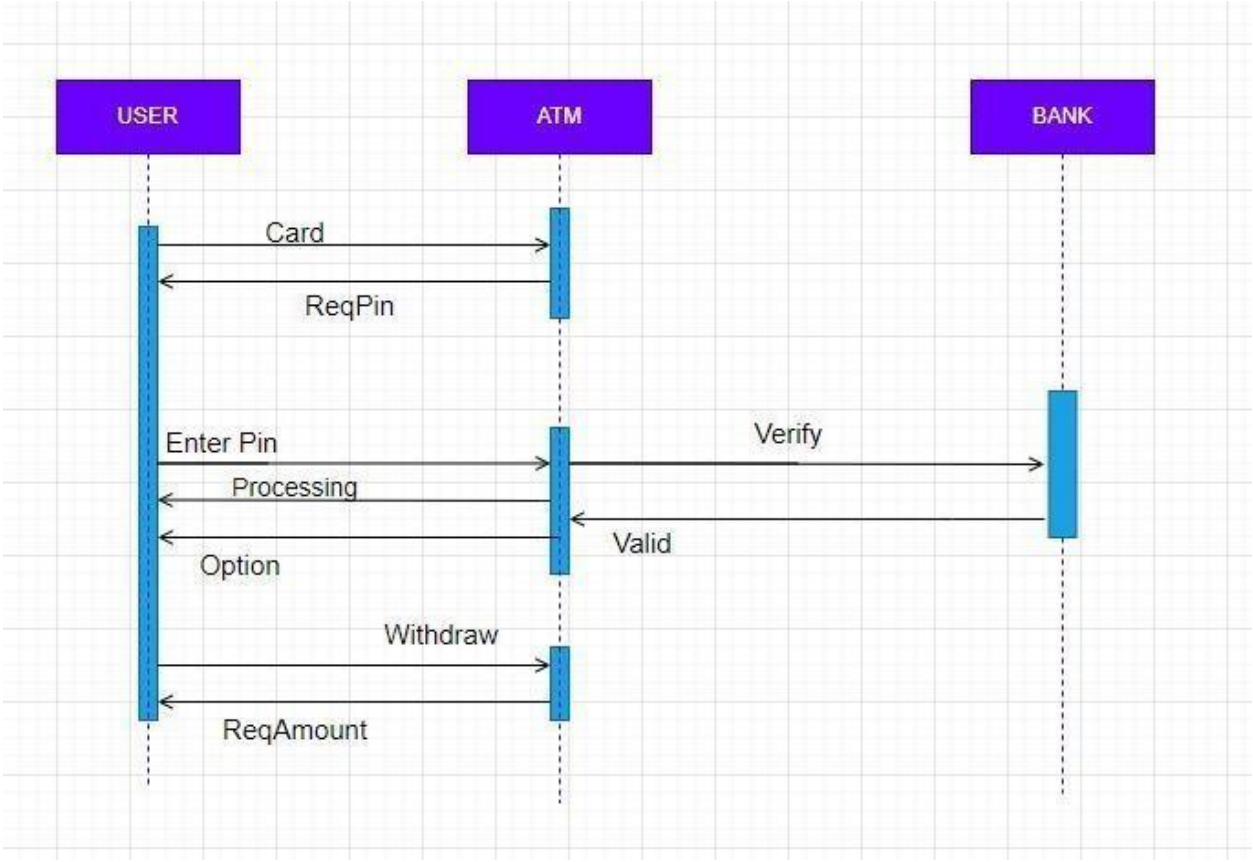


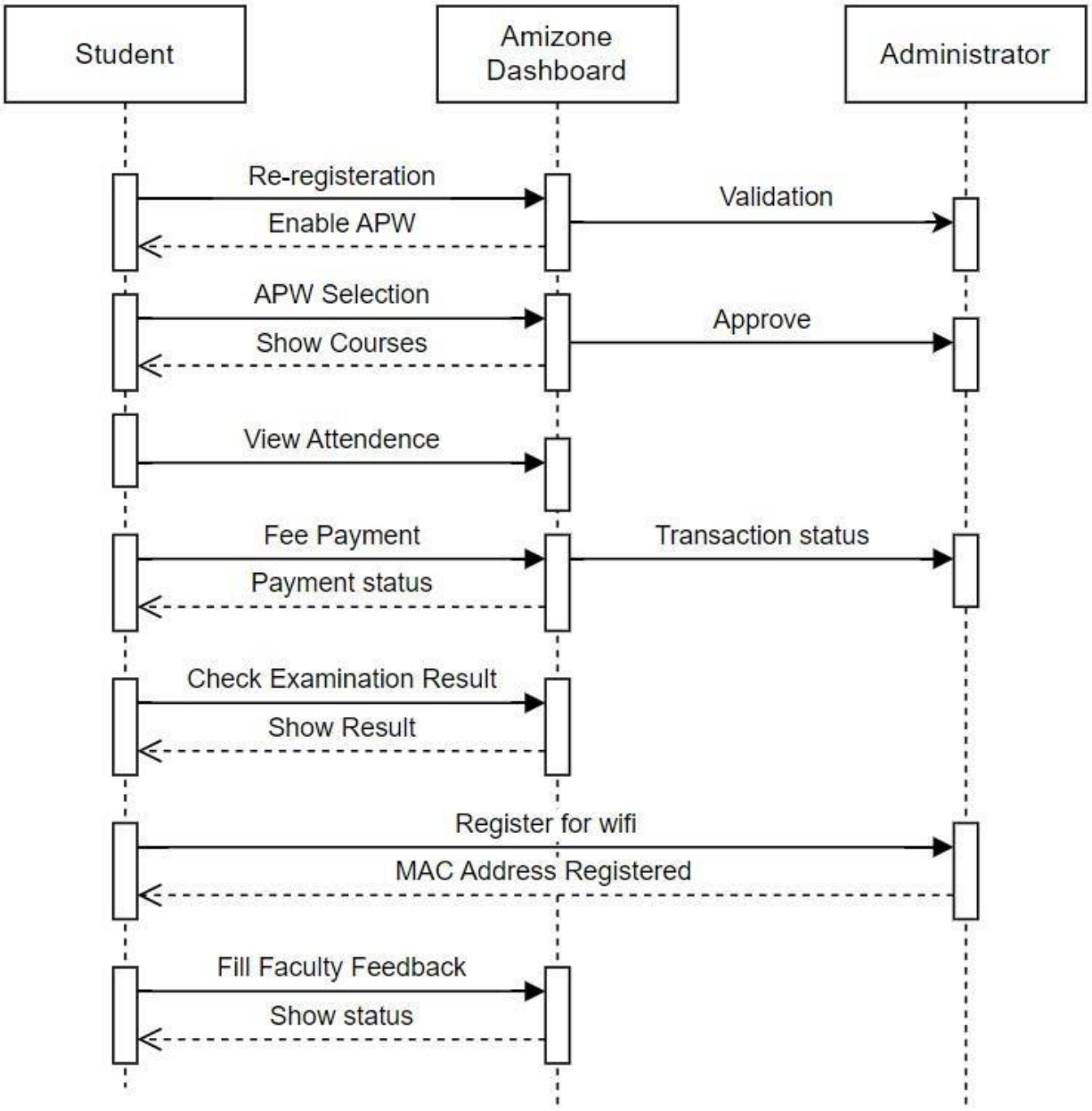
DIAGRAM:


USER
ATM
BANK
Card
ReqPin
Enter Pin
Verify
Processing
Valid
Option
Withdraw
ReqAmount

EXPERIMENT 4

AIM: Draw the sequence diagram of Amizone.

DIAGRAM:



EXPERIMENT – 5

AIM: Draw Activity Diagram of an ATM.

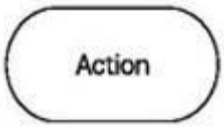
THEORY:

Activity diagram is another important diagram in UML to describe dynamic aspects of the system. Activity diagram is basically a flow chart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. So, the control flow is drawn from one operation to another.

Activity Diagram Annotations:

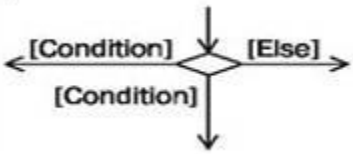
- **Initial Node:** The initial node is the starting point of an activity. An activity can have more than one initial node. It is also possible that an activity has no initial node but is initiated by an event (action: accepting an event).

- **Action:** An action is an individual step within an activity, for example, a calculation step that is not deconstructed any further. That does not necessarily mean that the action cannot be subdivided in the real world, but in this diagram will not be refined any further. The action can possess input and output information. The output of one action can be the input of a subsequent action within an activity. Specific actions are calling other actions, receiving an event, and sending signals.



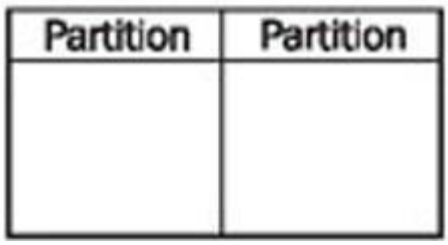
- **Edge:** Edges, represented by arrows, connect the individual components of activity diagram and illustrate the control flow of the activity: Within the control flow an incoming arrow starts a single step of an activity; after the step is completed the flow continues along the outgoing arrow. A name can be attached to an edge (close to the arrow).

- **Decision Node:** The diamond below represents a conditional branch point or decision node. A decision node has one input and two or more outputs.



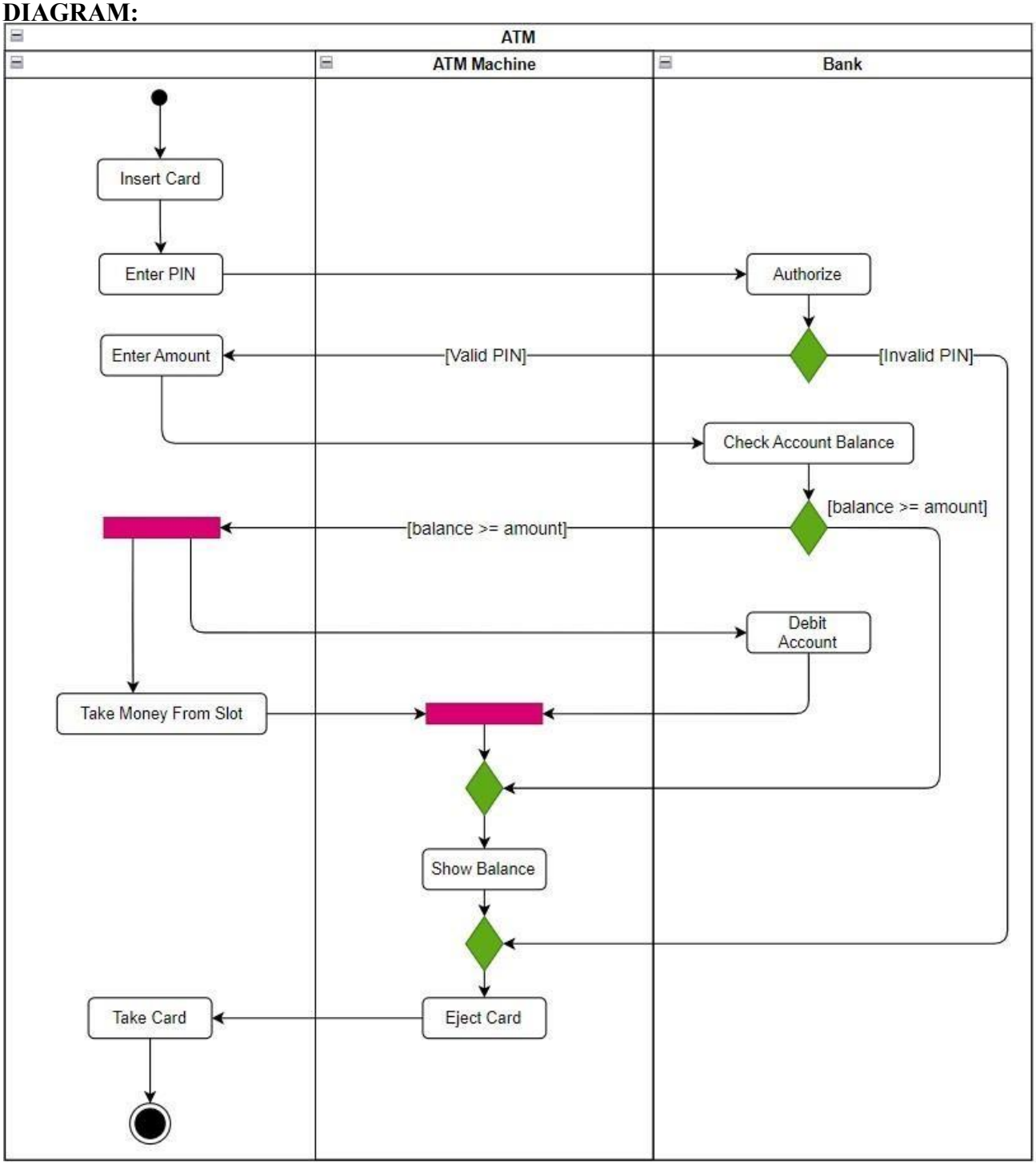
- **Activity Partition:** The individual elements of an activity diagram can be divided into individual areas or 'partitions'. Various criteria can lead to the creation of these partitions: organization entities, cost centers, locations, etc.
  Individual steps of an activity will be assigned to these partitions. Each partition is set apart from its neighboring partition by a horizontal or vertical continuous line; from this stems the

term swim lanes. Each partition receives a name. Partitions can be arranged in a twodimensional manner.



- **Activity Final Node**: The activity final node indicates that an activity is completed. An activity diagram can have more than one exit in the form of activity final nodes: If several parallel

DIAGRAM:

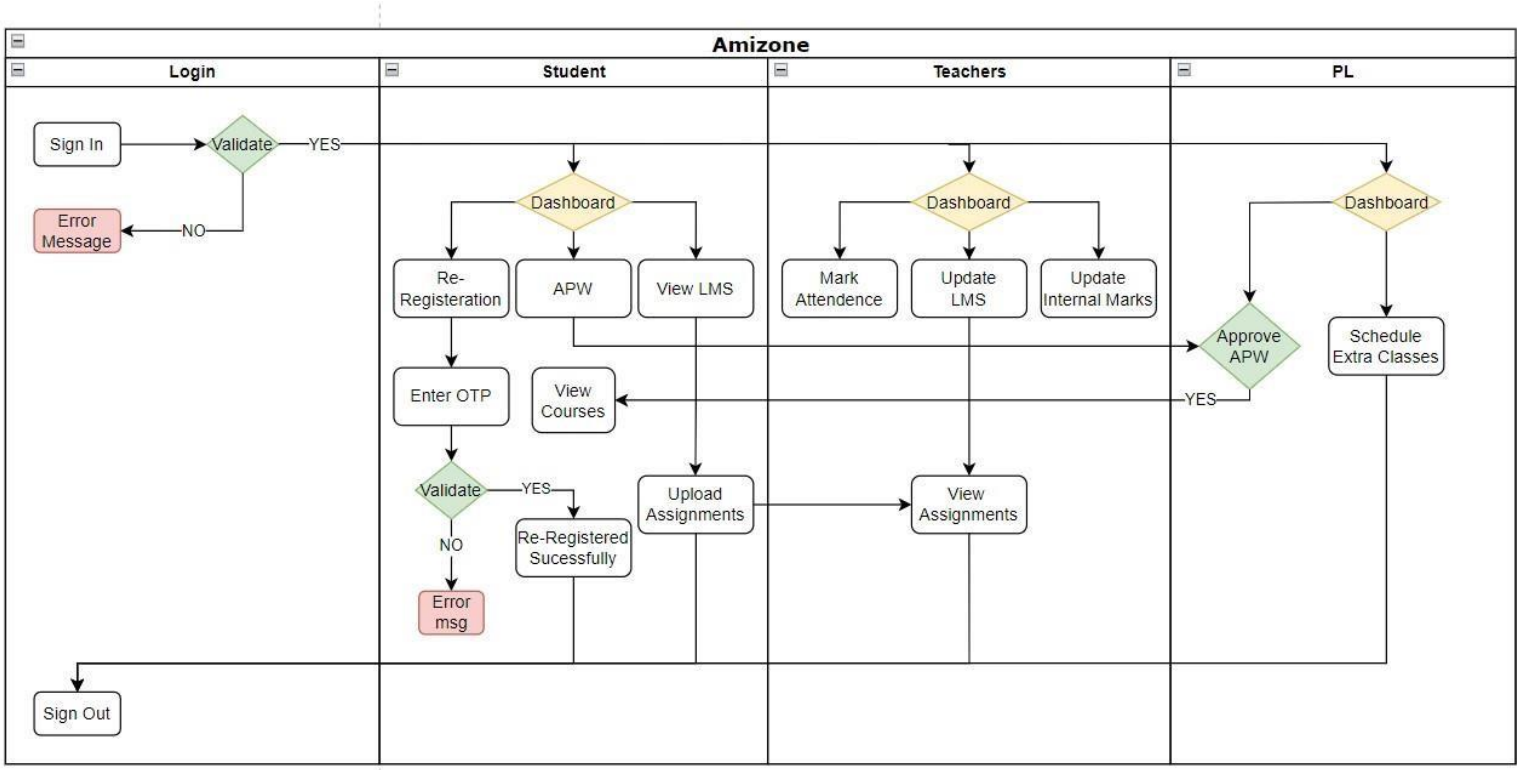


flows are present within an activity, all flows are stopped at the time the activity final node is reached.

EXPERIMENT – 6

AIM: Draw activity Diagram of Amizone.

DIAGRAM:



EXPERIMENT – 9

AIM: Draw the state chart diagram for an ATM machine.

THEORY:

A state chart diagram, also known as a state machine diagram, is a type of behavioral diagram in the Unified Modeling Language (UML). It models the behavior of individual objects or entities within a system by depicting the various states that the object can be in and the transitions between those states in response to events or actions.

State Chart Annotations:

- **State:** A state represents a condition or situation in which an object or entity exists at a particular point in time. It describes the behavior of the object when it is in that state. States are typically depicted as rounded rectangles with the state name inside.

- **Initial State:** The initial state represents the starting point of the state machine. It indicates the state of the object when it is first created or initialized. It is denoted by a solid circle attached to the starting state.

- **Final State:** The final state represents the end point of the state machine. It indicates the termination or completion of the object's behavior. It is denoted by a solid circle with a dot inside, attached to the state(s) from which the object can terminate.

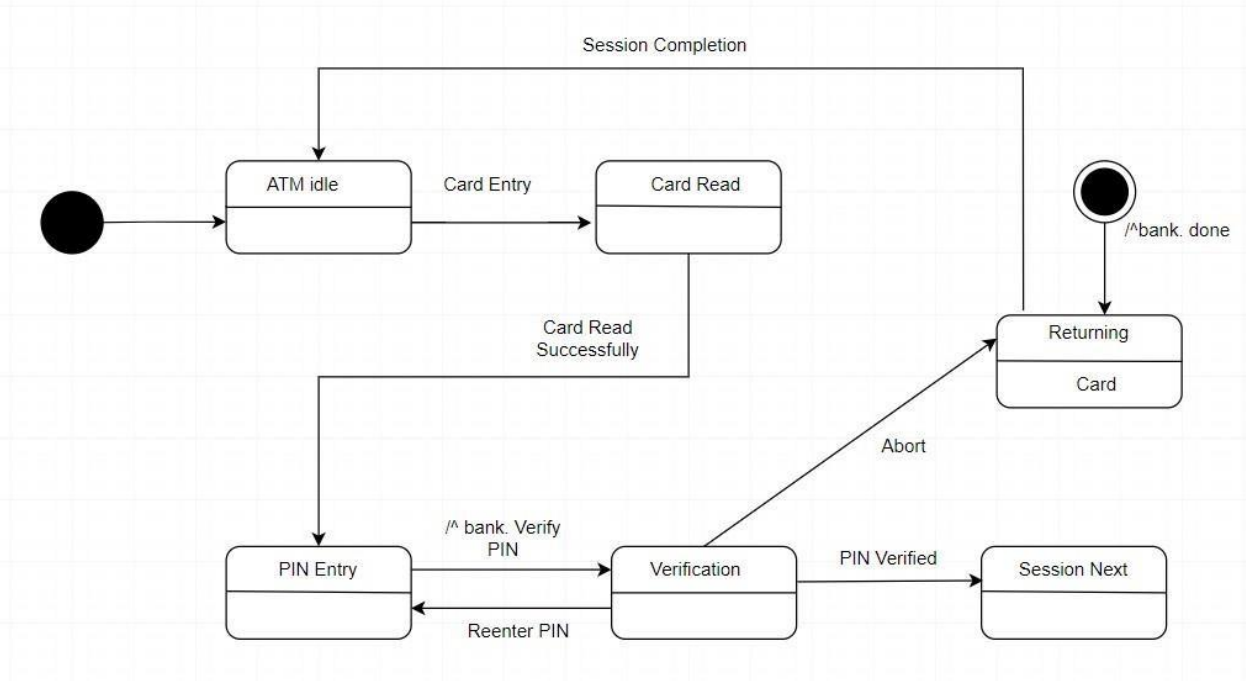
- **Transitions:** Transitions represent the movement of an object from one state to another in response to an event or action. They depict the conditions under which the transition occurs and the actions that are performed during the transition. Transitions are typically depicted as arrows between states, labeled with the triggering event or action and any associated conditions or actions.

- **Events:** Events are external stimuli or occurrences that trigger state transitions. They represent things that happen to the object from the environment or other parts of the system. Events can be triggered by user actions, system events, or the passage of time.

- **Actions:** Actions are behaviors or operations that are performed when a transition occurs. They represent the activities that the object performs as it moves between states. Actions can include computations, updates to variables, or interactions with other objects.

State chart diagrams are commonly used to model the dynamic behavior of objects or entities in systems with complex state-based logic. They provide a visual representation of the object's states and transitions, helping stakeholders understand how the object behaves under different conditions and how it responds to various events or actions. State chart diagrams are particularly useful for modeling the behavior of software systems, embedded systems, and other systems with stateful components.

DIAGRAM:


Session Completion
ATM idle
Card Entry
Card Read
/^bank. done
Card Read
Successfully
Returning
Card
Abort
/^ bank. Verify
PIN
PIN Entry
Verification
PIN Verified
Session Next
Reenter PIN

EXPERIMENT – 10

AIM: Draw the collaboration diagram of ATM

THEORY:

The UML Collaboration Diagram is used to show the relationship between the objects in a system. Instead of showing the flow of messages, a collaboration diagram depicts the architecture of the object residing in the system as it is based on object-oriented programming. The UML Collaboration diagrams are used by designers to define and clarify the roles of the objects that perform a particular flow of events of a use case. The UML Collaboration diagram is the primary source of information used to determine class responsibilities and interfaces.

Collaboration Diagram Annotations:

- **Objects:** Objects represent instances of classes or components within the system. Each object is depicted as a rectangle with the object name inside. Objects interact with each other by exchanging messages.

- **Links:** Links represent the relationships or associations between objects. They are typically represented by lines connecting objects, with optional labels to indicate the nature of the relationship.

- **Messages:** Messages represent the communication or interaction between objects. They depict the flow of information or control between objects. Messages can be synchronous (indicated by a solid arrow), asynchronous (indicated by a dashed arrow), or create messages (indicated by an arrow with a filled circle at the receiving end).

- **Roles:** Roles define the specific responsibilities or behaviors that objects play within a collaboration. They provide a way to specify the context in which objects interact with each other. Roles are often represented by labels next to the objects.

- **Interaction Occurrences:** Interaction occurrences represent specific instances of interactions between objects. They are used to depict the sequence of messages exchanged during a particular scenario or use case.

ADVANTAGES OF COLLABORATION DIAGRAM

1. The collaboration diagram is also known as Communication Diagram.
2. It mainly puts emphasis on the structural aspect of an interaction diagram, i.e., how lifelines are connected.
3. The syntax of a collaboration diagram is like the sequence diagram; the difference is that the lifeline does not consist of tails.
4. The messages transmitted over sequencing is represented by numbering each individual message.
5. The collaboration diagram is semantically weak in comparison to the sequence diagram.

6. The special case of a collaboration diagram is the object diagram. 7. It focuses on the elements and not the message flow, like sequence diagrams.

DIAGRAM: