INDEX

Practical No.	Practical Name	Date	Sign
1.	Study and implementation of class diagrams		
2.	Study and implementation of Use Case Diagrams		
3.	Study and implementation of Entity Relationship Diagram.		
4.	Study and implementation of Sequence Diagrams.		
5.	Study and implementation of State Transition Diagrams.		
6.	Study and implementation of Data Flow Diagrams.		
7.	Study and implementation of Collaboration Diagrams.		
8.	Study and implementation of Activity Diagrams.		
9.	Study and implementation of Component Diagrams.		
10.	Study and implementation of Deployment Diagrams.		

Practical 1: Class Diagram

Study and implementation of class diagram

A class diagram in UML (Unified Modeling Language) is a type of structural diagram that represents the structure of a system or software application by modeling its classes, attributes, methods, and their relationships. Class diagrams are a fundamental part of UML and are widely used in software engineering for visualizing and designing software systems. Class diagrams in UML are an essential tool for designing and documenting the structure of a software system, helping developers and stakeholders understand the relationships between classes and how they collaborate to achieve the system's functionality.

The key elements and concepts in a class diagram:

1. **Class:** The central element of a class diagram is the class itself. A class represents a blueprint or template for creating objects. It typically consists of a name, attributes (data members), and operations (methods).
2. **Attributes:** Attributes are data members or properties of a class. They represent the state of an object and are shown as name: type pairs, where the name is the attribute's name, and the type is its data type. For example, `name: String` indicates an attribute named "name" with a data type of String.
3. **Operations:** Operations, also known as methods, are the behaviors or functions that a class can perform. They are depicted as name (parameters): return type. For instance, getName(): String represents an operation named "getName" that takes no parameters and returns a String.
4. **Visibility:** Each attribute and operation can have a visibility indicator, which indicates the access level of the member. The common visibility levels in UML are:

- **+ (public):** The member is accessible from anywhere.
- **- (private):** The member is accessible only within the class.
- **#` (protected):** The member is accessible within the class and its subclasses. (package): The member is accessible within the package or namespace.

5. Association: Associations represent relationships between classes. They show how classes are connected or related to each other. Associations can have multiplicity (e.g., 0...1, 1, 0..*) to indicate how many instances of one class are associated with instances of another class.

6. Aggregation: Aggregation is a special type of association that represents a "whole- part" relationship between classes. It is denoted by a diamond shape on the side of the whole class. For example, a Car class may have a "has" relationship with a Wheel class.

7. Composition: Composition is a stronger form of aggregation, indicating that the "part" class cannot exist without the "whole" class. It is represented by a filled diamond shape on the side of the whole class.

8. Inheritance/Generalization: Inheritance represents an "is-a" relationship between classes, where one class (subclass or derived class) inherits the attributes and operations of another class (super class or base class). It is represented by an arrow pointing from the subclass to the super class.

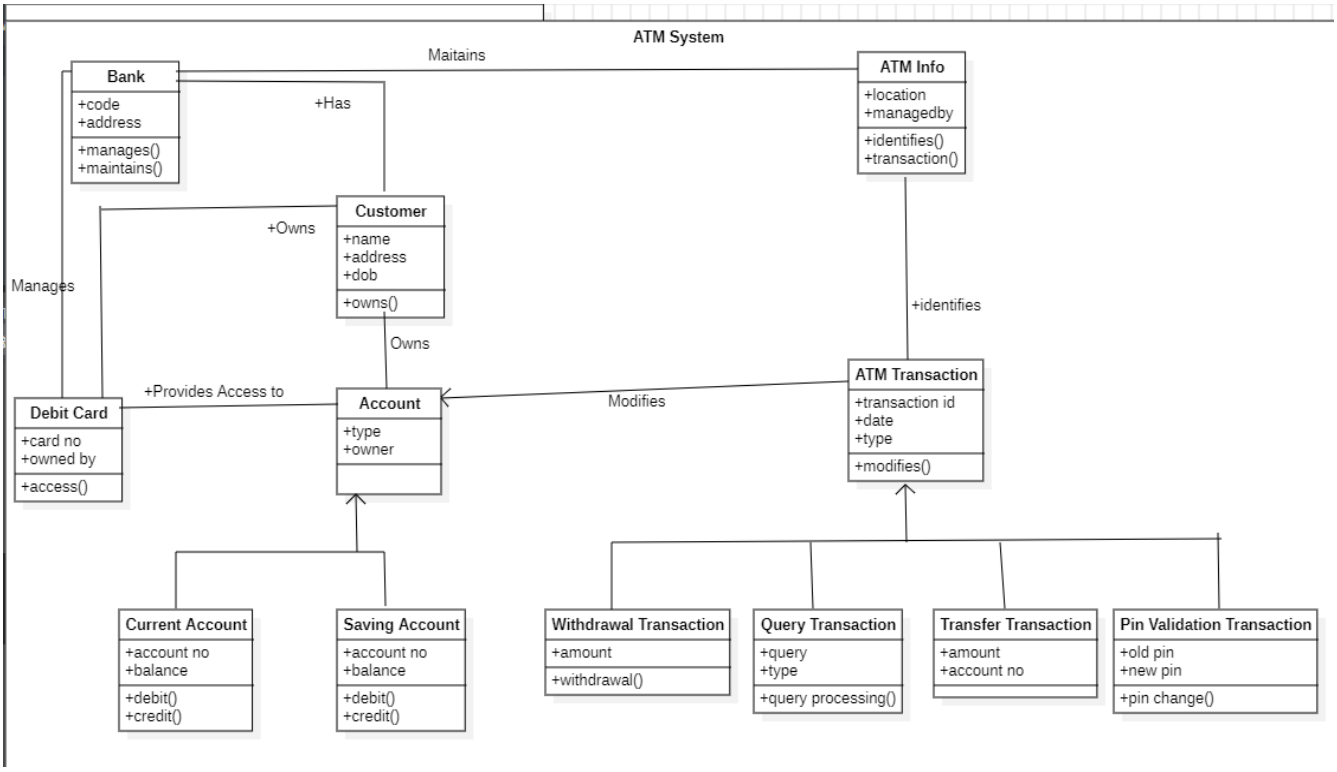
9. Interface: Interfaces represent a contract that a class must adhere to by implementing its methods. Interfaces are denoted with a <<interface>> stereotype, and classes that implement an interface are connected to it with a dashed line.

10. Dependency: Dependency is a relationship where one class depends on another class, often in terms of method parameter types or return types. It is represented by a dashed arrow.

11. Multiplicity: Multiplicity is used to specify the number of instances that can participate in an association. It is shown as a range (e.g., 0..1, 1, 0.."), indicating the minimum and maximum number of instances allowed.

12. Stereotypes: Stereotypes are additional information or annotations that can be added to classes, associations, or other UML elements to provide additional context or meaning. Class

Diagram for ATM System:



Practical 2: Use Case Diagrams

Study and implementation of Use Case Diagrams.

A use case diagram in UML (Unified Modeling Language) is a type of behavioral diagram that represents the interactions between a system (and a part of a system) and external entities, known as actors. It is used to describe the functional requirements and behavior of a system from the perspective of its users. The key elements and concepts in a use case diagram:

1. Actor: Actors are external entities that interact with the system being modeled. Actors can represent various entities, such as users, other systems, or hardware devices. In a use case diagram, actors are depicted as stick figures. Actors are associated with use cases to show which use cases they are involved in.

2. Use Case: A use case represents a specific functionality or behavior of the system from the user's perspective. It describes a sequence of interactions between the system and its actors to achieve a specific goal. Use cases are depicted as ovals with a name inside. They represent a unit of functionality that provides value to the actors.

3. Association: Associations are lines connecting actors and use cases to indicate that an actor interacts with a particular use case. An association arrow points from the actor to the use case, showing the direction of interaction.

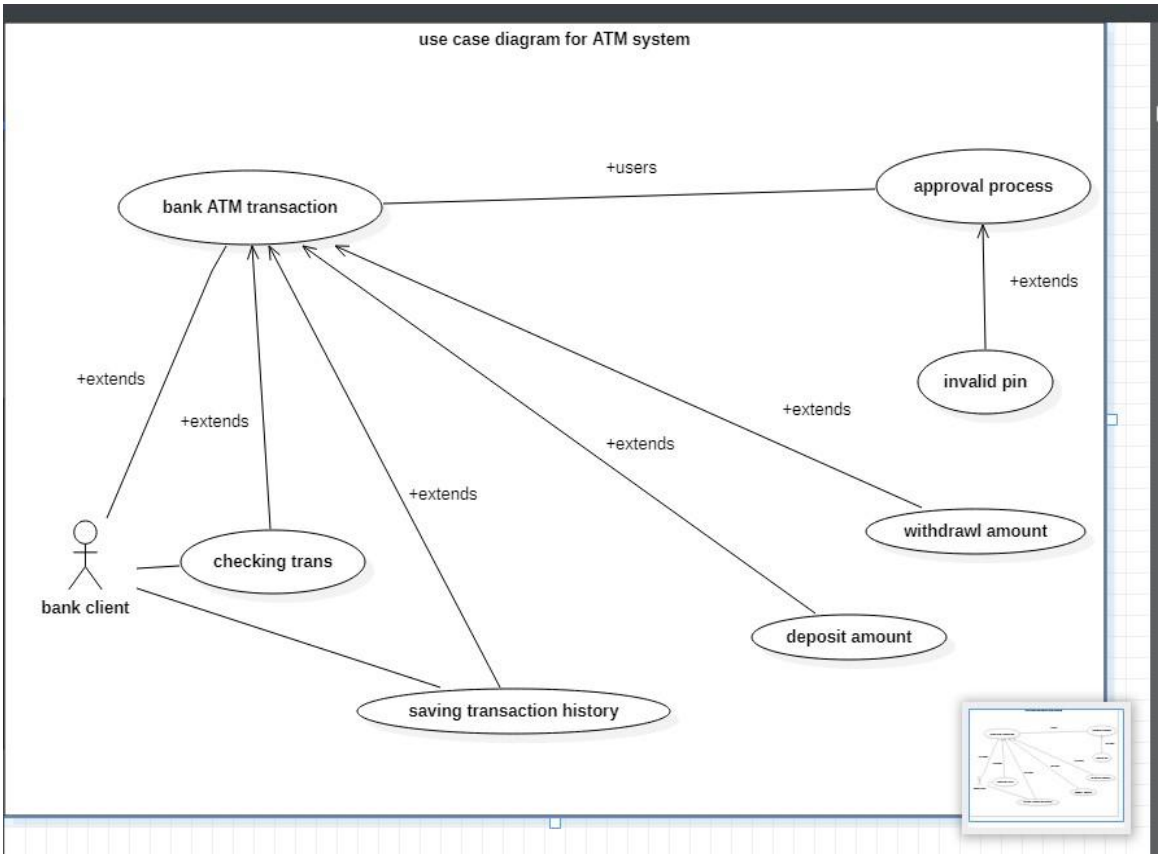
4. System Boundary: A use case diagram often includes a system boundary, which is a box or boundary that encloses all the use cases and actors. This boundary represents the scope of the system being modeled.

5. Extend Relationship: An extend relationship is used to show optional or conditional behavior within a use case. It is represented by a dashed arrow with the "extend>" stereotype. The extending use case is triggered based on certain conditions when the base use case is executed.

6. Include Relationship: An include relationship is used to show that one use case includes another use case as a part of its behavior. It is represented by a dashed arrow with the "«<include>>" stereotype. The included use case is always executed as part of the base use case.

7. Generalization/Inheritance: Generalization in a use case diagram represents an inheritance relationship between use cases. It is similar to the inheritance concept in class diagrams. A child use case inherits the behavior of a parent use case and may extend or specialize it. It is represented by an arrow with an open triangle pointing from the child to the parent use case.

Use Diagram For ATM System:



Practical 4: Sequence Diagram

Study and implementation of Sequence Diagrams.

A sequence diagram in UML (Unified Modeling Language) is a type of behavioral diagram that illustrates the interactions and sequences of messages or calls between objects or components within a system over time. Sequence diagrams are particularly useful for modeling the dynamic aspects of a system, including how objects collaborate and communicate to accomplish specific tasks or scenarios. The key elements and concepts in a sequence diagram:

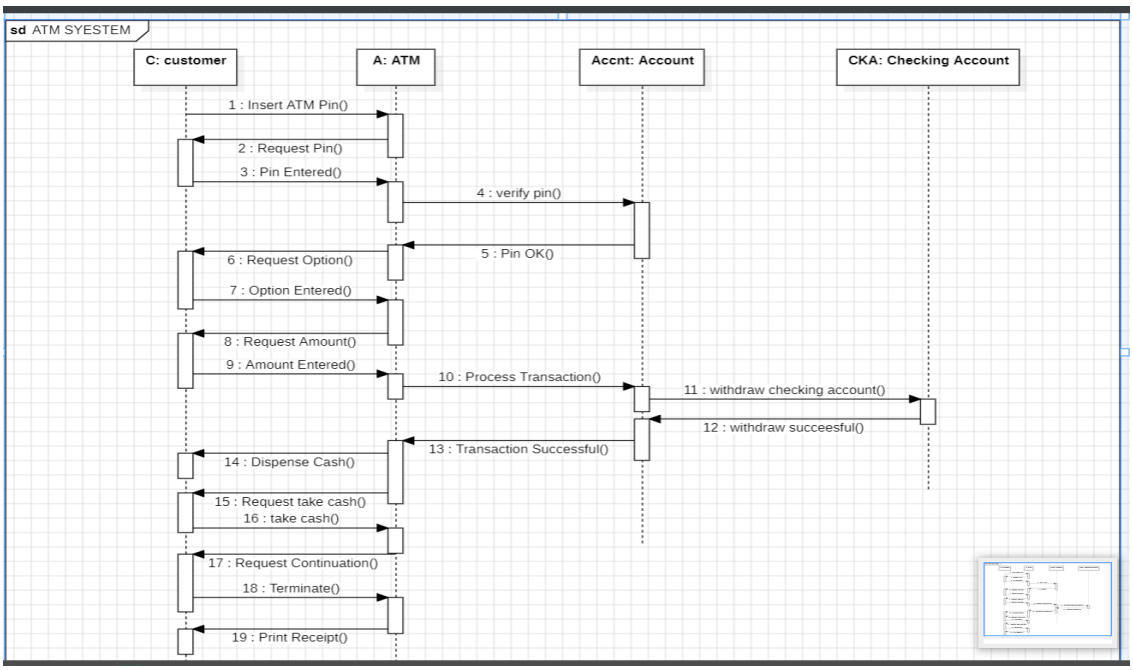
1. Lifeline: A lifeline represents an object or entity involved in the sequence of interactions. It is depicted as a vertical dashed line that extends from the top to the bottom of the diagram. Each lifeline has a name (usually the name of the object it represents) and can have a stereotype (e.g., actor, boundary, control) to indicate its role.

2. Activation Bar: An activation bar, also known as an execution occurrence or activation rectangle, represents the period during which an object is actively processing or executing a task. It is typically shown as a horizontal bar located on a lifeline and is drawn from left to right, indicating the duration of the object's activity.

3. Message: A message represents a communication or interaction between two objects. Messages can be synchronous (blocking), asynchronous (non-blocking), or a return message. They are represented by arrows connecting the lifelines of the sending and receiving objects. Messages can have labels to describe the type or purpose of the communication.

4. Self-Message: A self-message is a message that an object sends to itself. It is depicted as a loopback arrow from the lifeline of the sender back to itself.

Sequence Diagram for ATM System:



Practical 3: ER Relationship Diagrams

Study and implementation of Entity Relationship Diagrams.

An Entity-Relationship (ER) diagram is a visual representation of the data model for a database system. ER diagrams are widely used in database design and modeling to depict the structure of a database, including the entities (objects or concepts), their attributes) (properties), and the relationships between them. The key elements and concepts in an ER diagram:

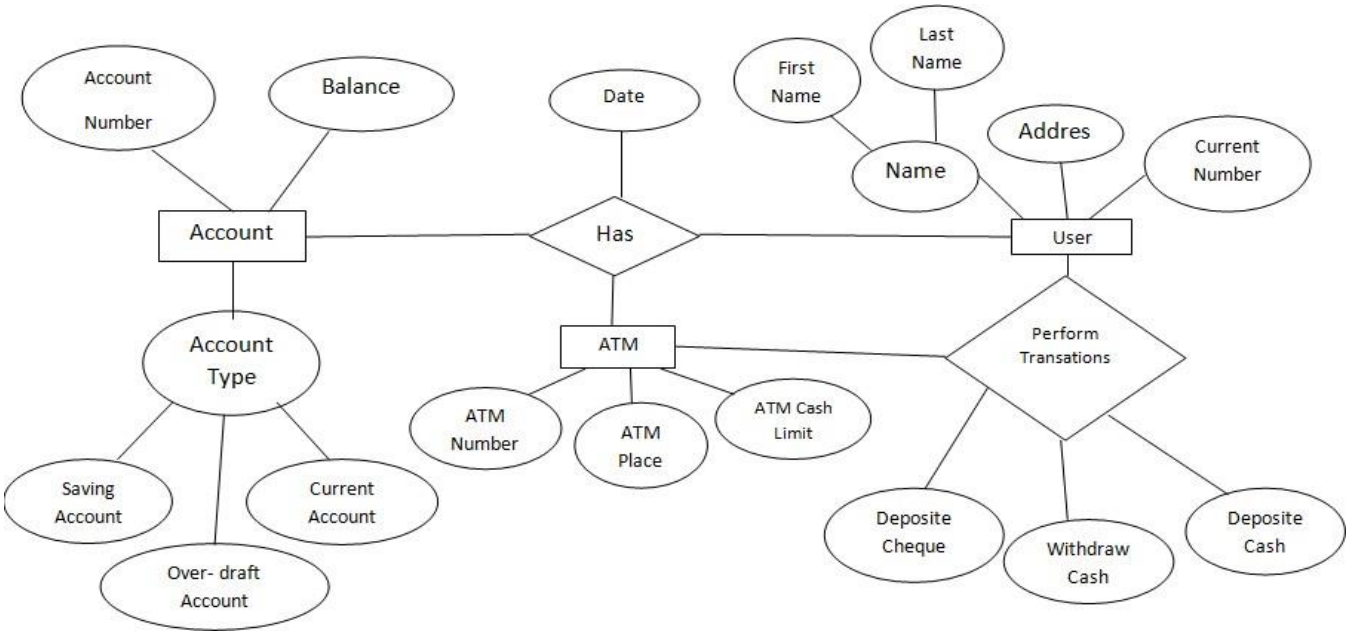
1. **Entity:** An entity is a real-world object, concept, or thing that has a distinct existence and can be uniquely identified. In an ER diagram, entities are represented as rectangles. Each entity is typically labeled with its name, and it can have various attributes

2. **Attributes:** Attributes are properties or characteristics that describe entities. Attributes are depicted as ovals connected to their respective entities. Each attribute has a name and a data type that specifies the kind of data it can hold (e.g., integer, string, date).

3. **Relationship:** A relationship represents an association between two or more entities. It describes how entities are related to each other. Relationships are depicted as diamond shapes connecting the participating entities. Each relationship has a name that describes the nature of the association.

4. **Cardinality:** Cardinality defines the number of instances of one entity that can be associated with the instances of another entity through a relationship.

ER Diagram for ATM System:



Practical 5: State Transition Diagrams

Study and implementation of State Transition Diagrams.

A state diagram, also known as a state machine diagram, is a type of behavioral diagram in UML (Unified Modeling Language) used to model the dynamic behavior of a system or an object by representing its various states and transitions between those states. State diagrams are particularly useful for visualizing the behavior of entities that exhibit different states and how they respond to events or stimuli. The key elements and concepts in a state diagram:

1. State: A state represents a condition or situation in which an object or system can exist. States are depicted as rounded rectangles with a label inside, indicating the name of the state. Each state can represent a specific mode, status, or phase of an object or system.

2. Initial State: An initial state, often denoted as a solid filled circle or a filled arrowhead, represents the starting point when the object or system is first created or enters a state machine. It indicates the state from which transitions can occur upon the activation of the state machine.

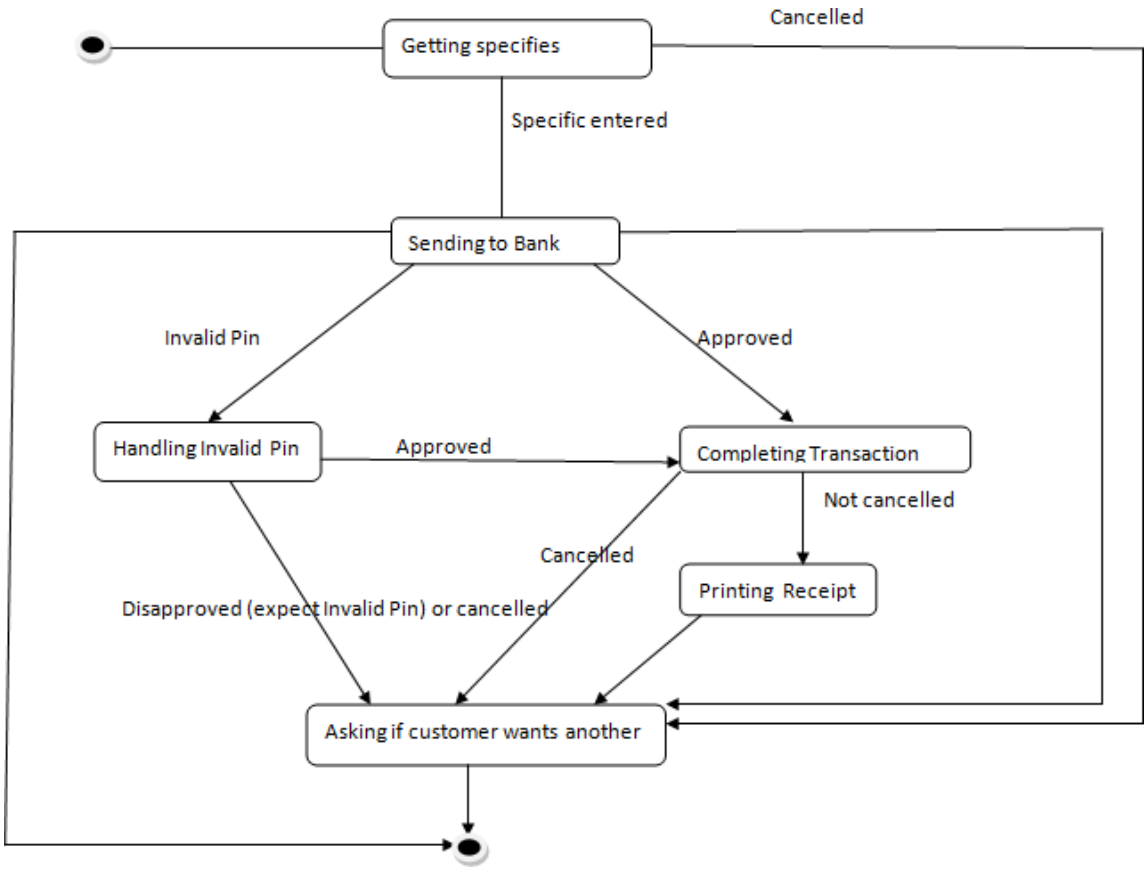
3. Final State: A final state, represented by a circle with a dot inside or the word "final," signifies the termination of the state machine or the completion of an object's lifecycle. When an object or system reaches a final state, it typically means that it has finished its task or operation.

4. Transition: A transition is a directed relationship between two states that specifies how an object or system changes from one state to another in response to an event or condition. Transitions are depicted as arrows with labels indicating the triggering event, guard conditions, and actions that occur during the transition. A transition arrow points from the source state to the target state.

5. Event: An event is an occurrence or stimulus that triggers a transition from one state to another. Events can be external, such as user inputs, signals, or timers, or internal, indicating that the object's internal conditions have changed. Events are usually labeled on transition arrows.

6. Guard Condition: A guard condition is a Boolean expression associated with a transition that determines whether the transition will be taken when the event occurs. It helps specify conditions under which a transition should occur or be deferred.

State Diagram for One Transaction ATM machine:



Disapproved
(expert Invalid Pin)

Practical:-6

Study and implementation of Data Flow Diagrams.

Data Flow Diagrams show information transfers and process steps of a system. The general concept is an approach of a depicting how occurs input in a system, further processes and what runs out. The aim of DFD is in accomplishing of understanding between developers and users, Data flow diagrams are maintained with other methods of structured systems analysis.

A data flow diagram (DFD) illustrates how data is processed by a system in terms of inputs and outputs. As its name indicates its focus is on the flow of information, where data comes from, where it goes and how it gets stored.

• Graphical representation of the "flow" of data through an information system;

• Modeling process aspects;

• An overview of the system;

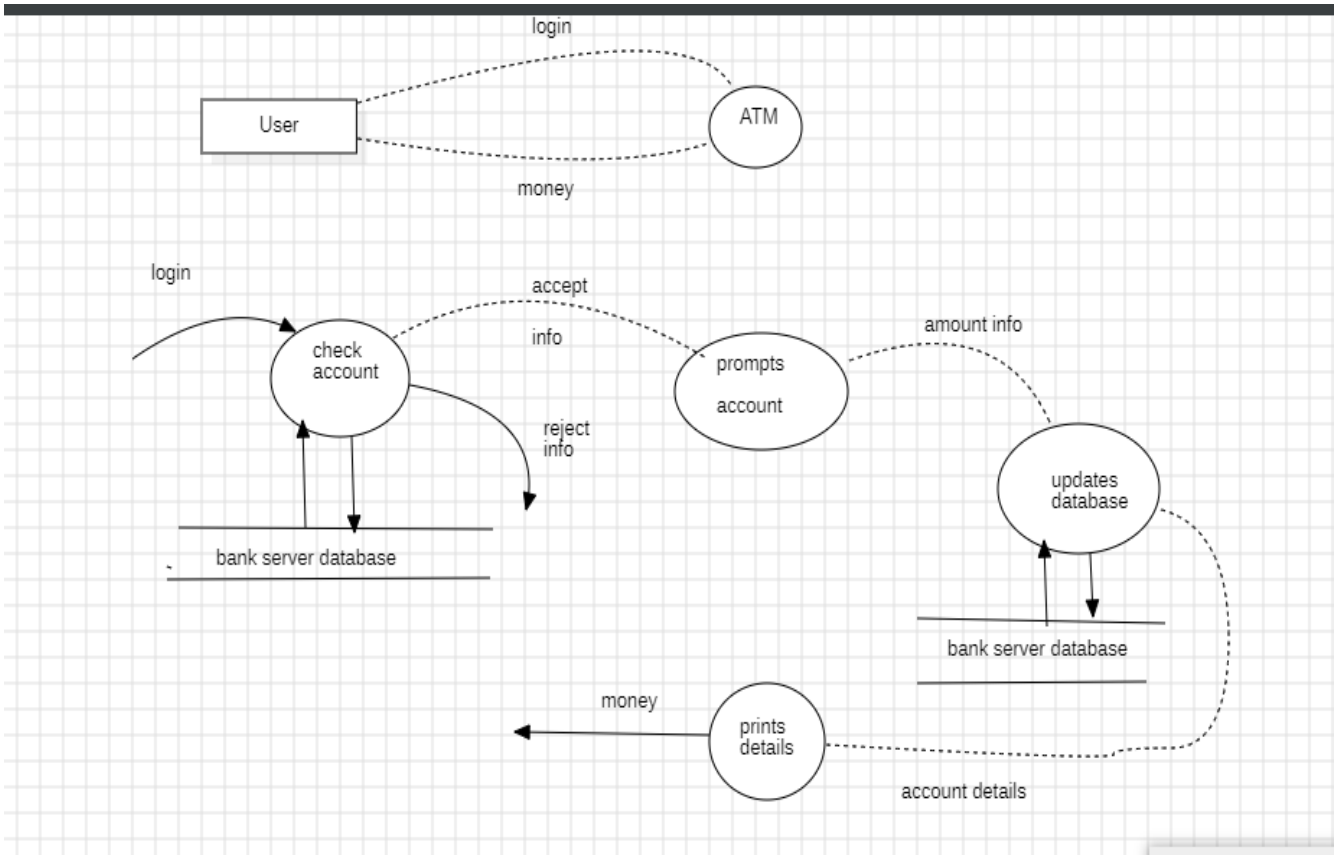
• For the visualization of data processing (structured design);

What kinds of information will be input to and output from the system;

Where the data will come from and go to;

• Where the data will be stored.

Data Flow Diagram For ATM System



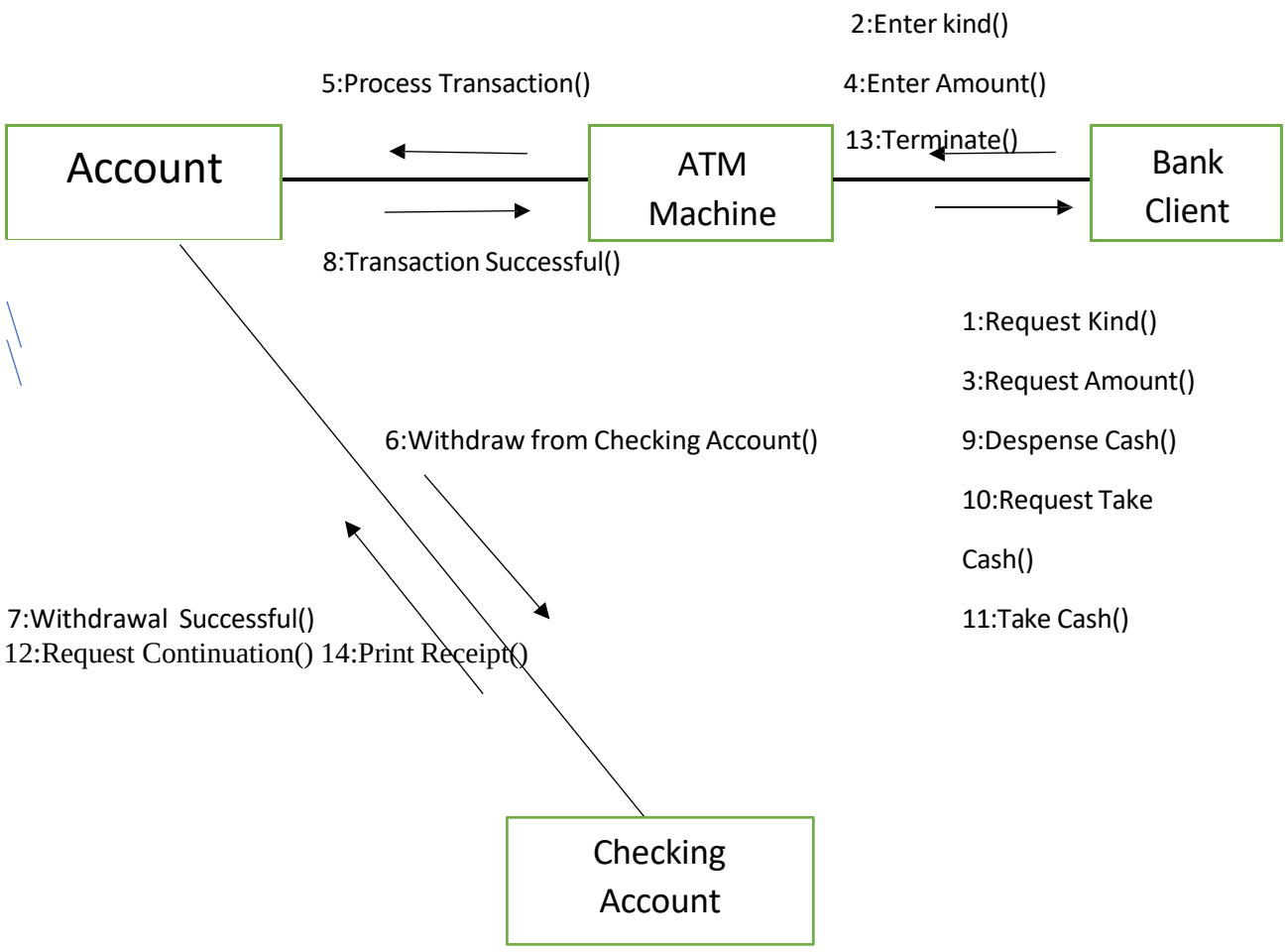
Practical 7: Collaboration Diagram

Study and implementation of Collaboration Diagrams.

Collaboration diagrams are used to show how objects interact to perform the behavior of a particular use case, or a part of a use case. Along with sequence diagrams, collaborations are used by designers to define and clarify the roles of the objects that perform a particular flow of events of a use case. They are the primary source of information used to determining class responsibilities and interfaces.

Unlike a sequence diagram, a collaboration diagram shows the relationships among the objects. Sequence diagrams and collaboration diagrams express similar information, but show it in different ways. Collaboration diagrams show the relationships among objects and are better for understanding all the effects on a given object and for procedural design.

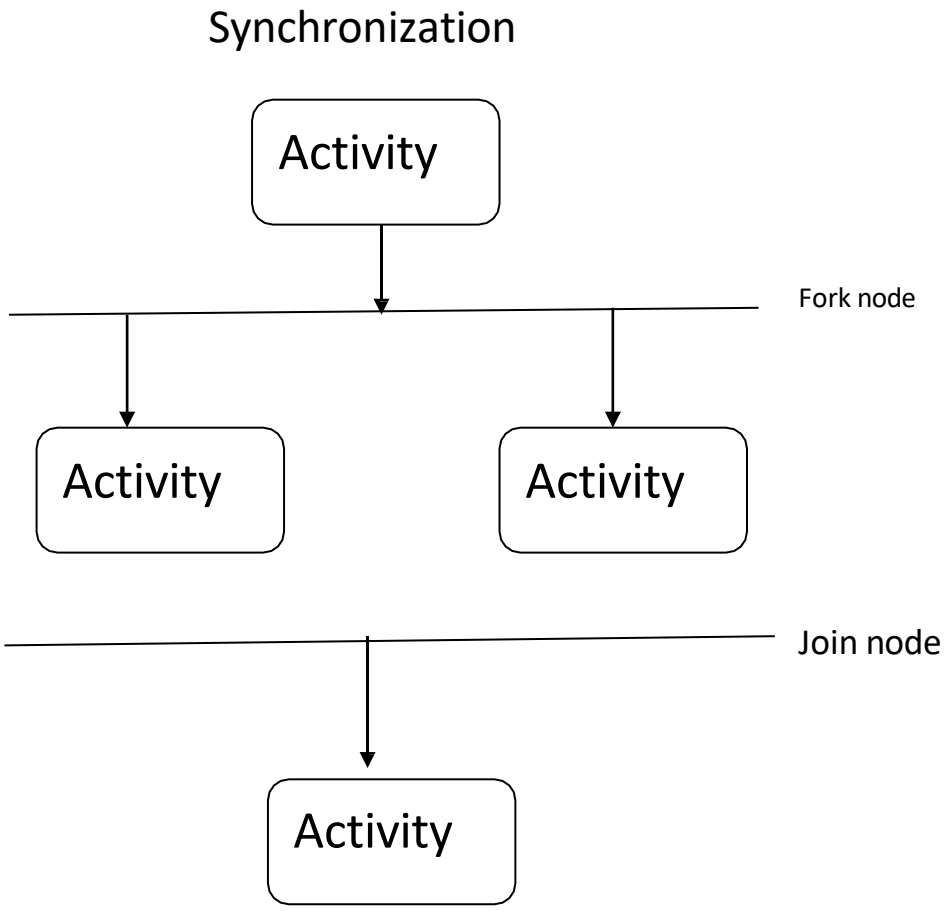
Collaboration Diagram For ATM System:



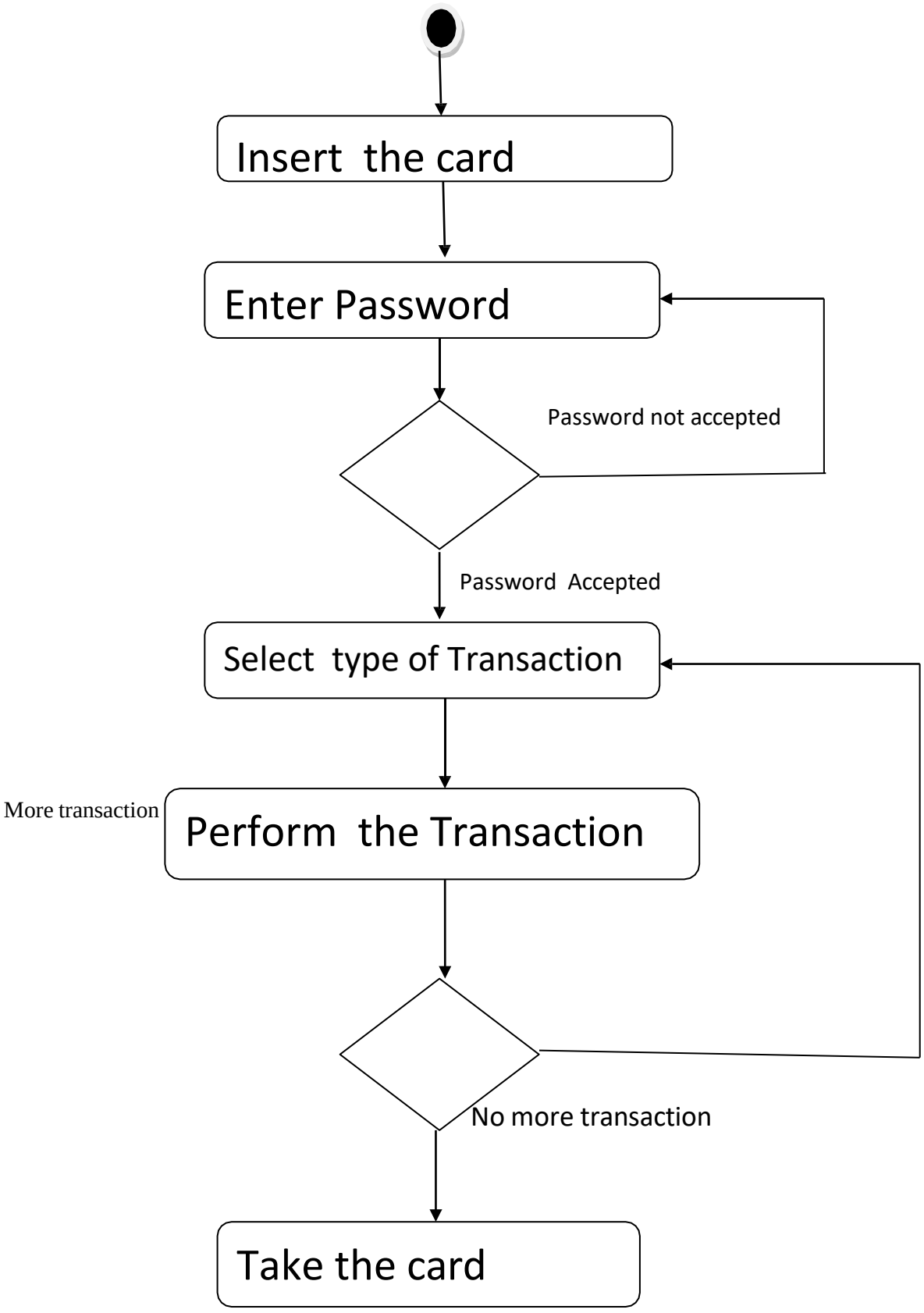
Practical 8: Activity Diagram

Study and implementation of Activity Diagram

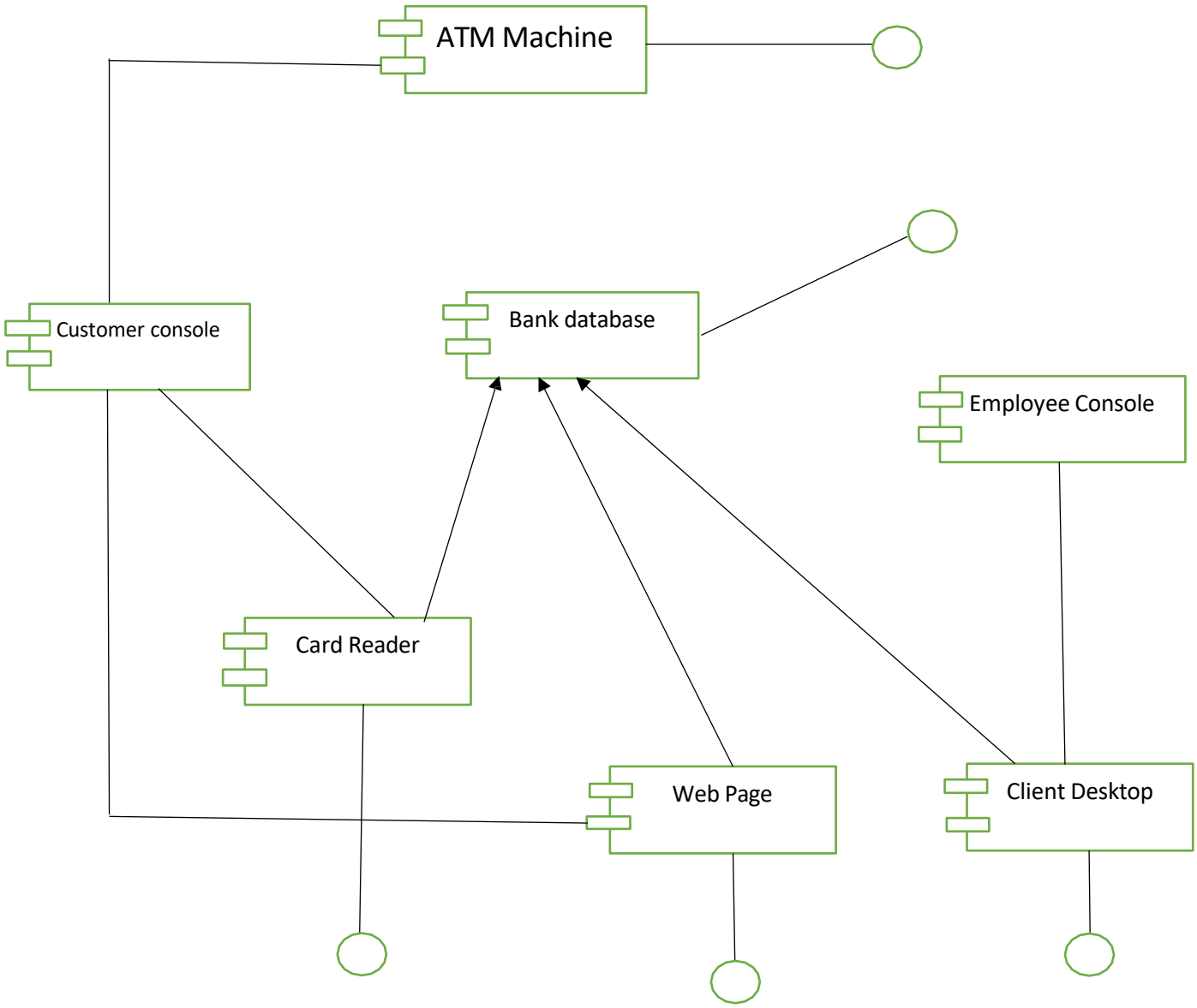
An activity diagram visually presents a series of actions or flow of control in a system similar to a flowchart or a data flow diagram. Activity diagrams are often used in business process modeling. They can also describe the steps in a use case diagram. Activities modeled can be sequential and concurrent.



Activity Diagram for Overall ATM Machine



Practical 9: Component Diagram



Practical: 10 Deployment Diagram

Study and implementation of Deployment Diagrams.

Deployment Diagram also 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.

A deployment diagram can cover a wide variety of levels within a single model. For instance, one diagram might cover how the application layers are connected, another might indicate where the source code is located within the system, and another might show how an executable is used across several nodes.

Deployment Diagram For ATM System