THAKUR TRUSTS

Thakur Educational Trust's (Regd.)
THAKUR RAMNARAYAN
COLLEGE OF ARTS & COMMERCE
ISO 21001:2018 Certified

trcac

SOFTWARE ENGINEERING JOURNAL

S.Y.B.Sc. – I.T. Semester - IV

Practical 1

Study and implementation of class diagrams.

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.

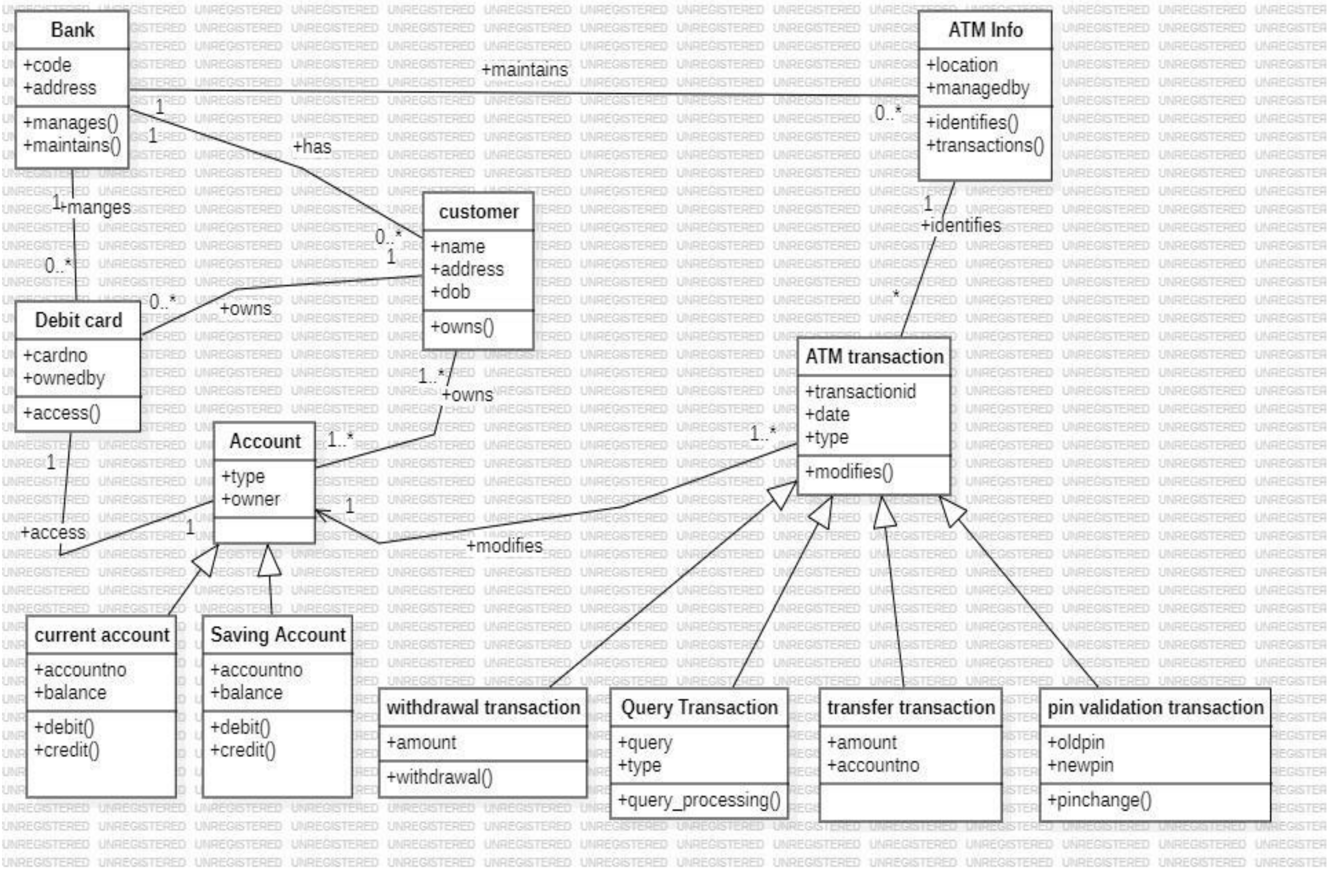
The key elements and concepts in a class diagram:

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. No Bank means ne Branches
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 (superclass or base class). It is represented by an arrow pointing from the subclass to the superclass.
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 Name
Attribute
Operations
Responsibilities

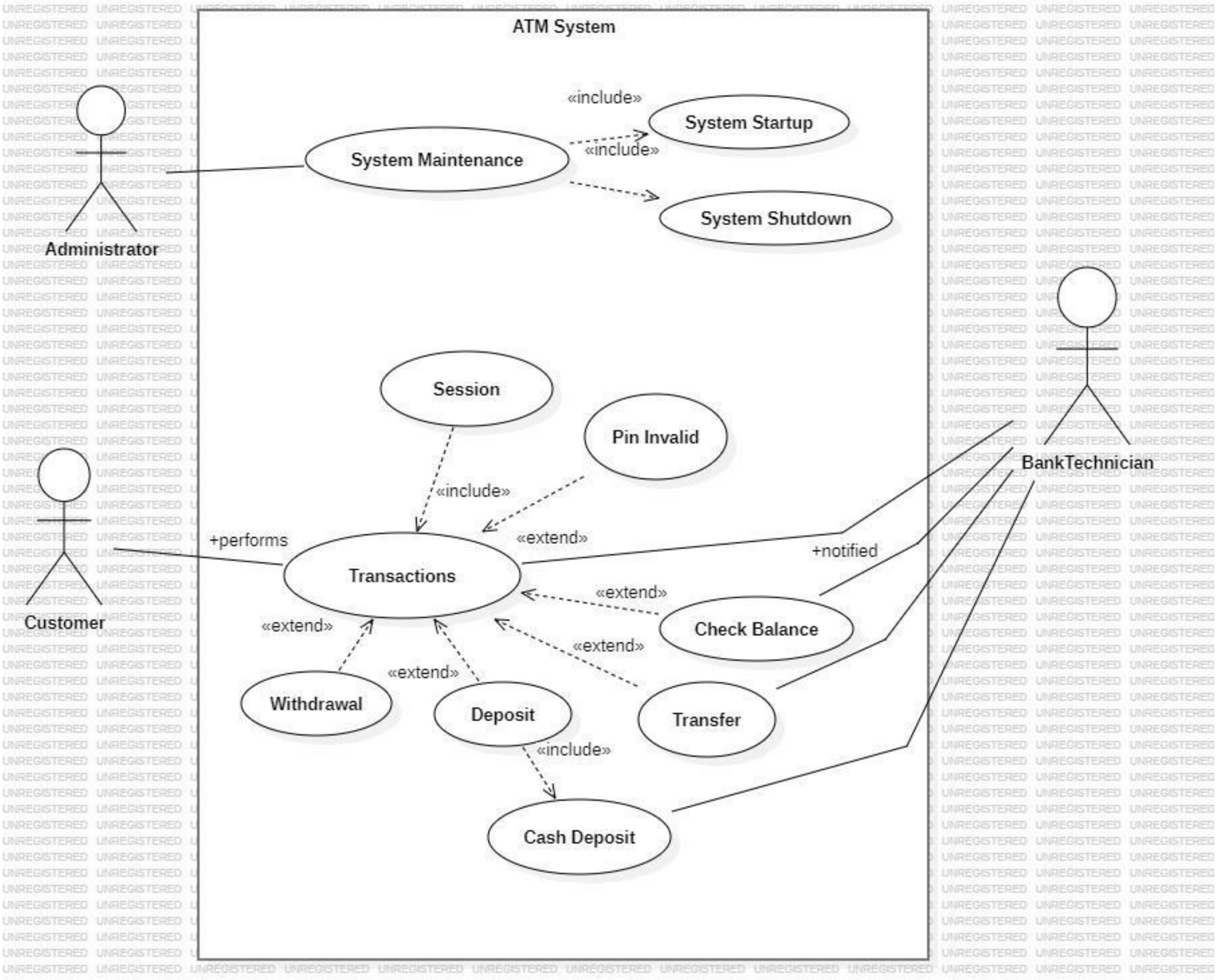


Practical 2

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 (or 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.



Practical 4

.

Study and implementation of Sequence Diagrams.

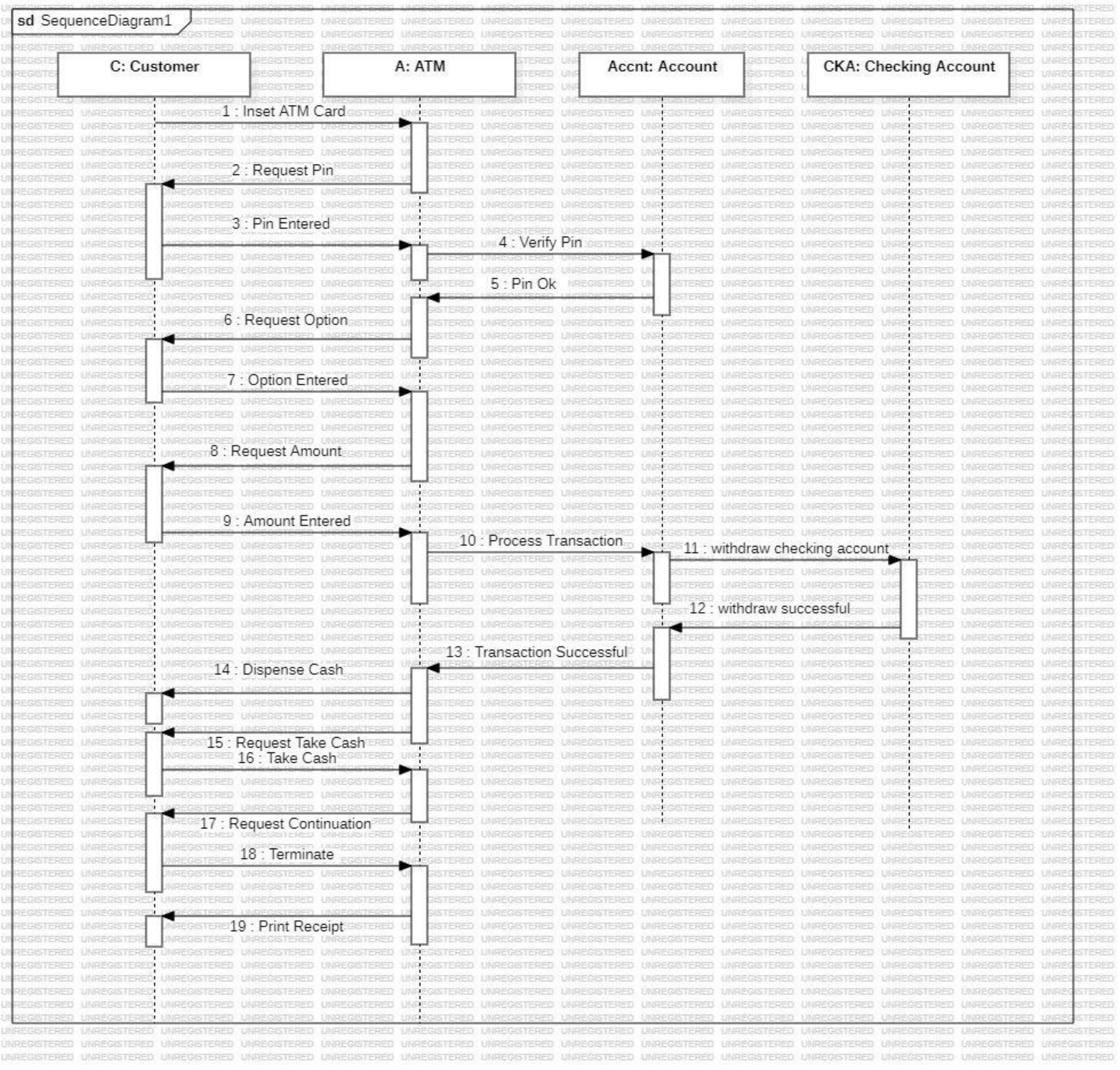
A sequence diagram in UML (Unified Modeling Language) is a type of behavioral diagram that illustrates the interactions and sequences of messages or calls between objects or components within a system over time. Sequence diagrams are particularly useful for modeling the dynamic aspects of a system, including 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.



Practical 5

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.

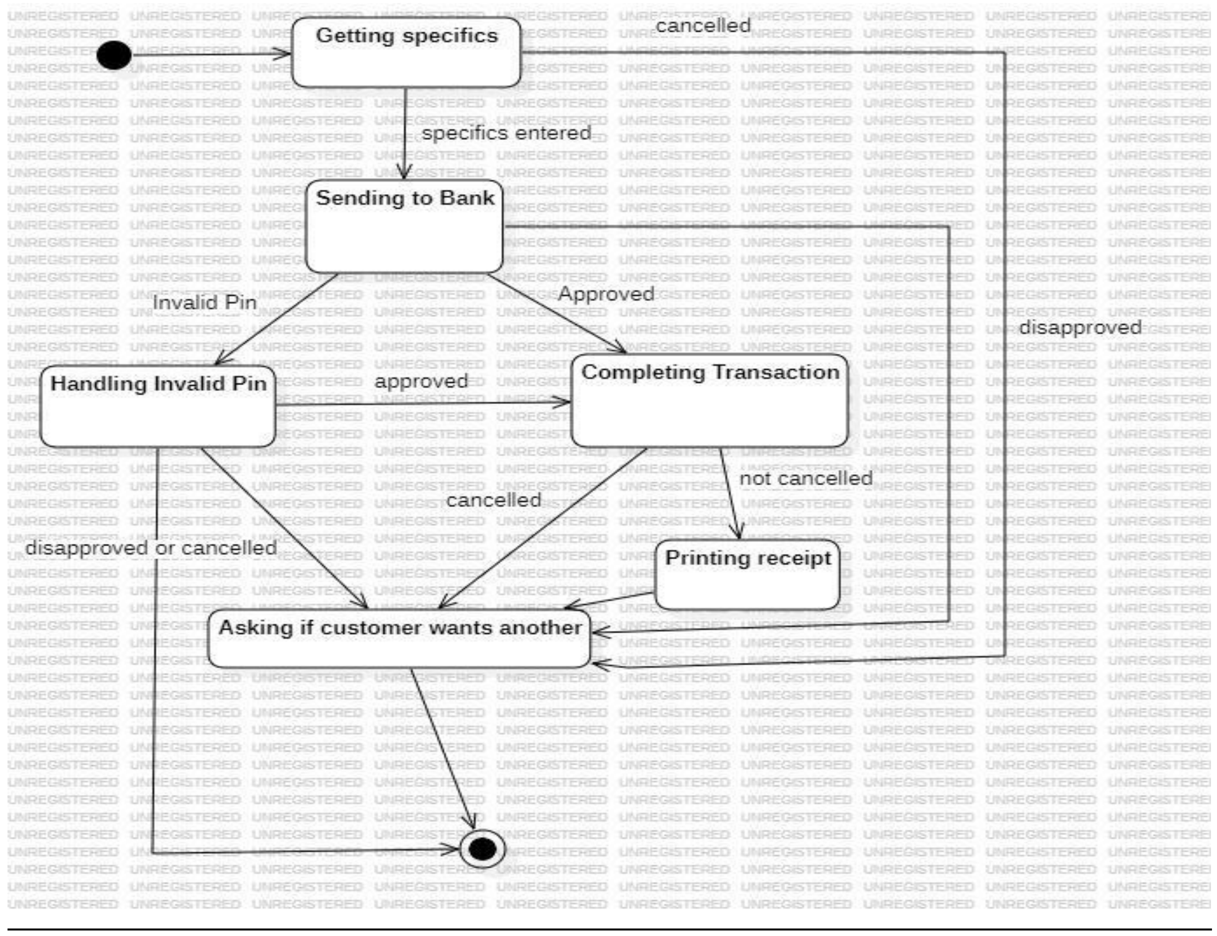
2. Initial State: An initial state, often denoted as a solid filled circle or a filled arrowhead, represents the starting point when the object or system is first created or enters a state machine. It indicates the state from which transitions can occur upon the activation of the state machine.

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 that it has finished its task or operation. state, it typically means

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.



Practical 8

Study and implementation of Activity Diagrams.

An activity diagram is a type of behavior diagram in UML(UNIFIED MODELING Language) that visually represents the flow of activities or actions within a system or a model the workflow, business processes, An activity diagram is a type of behavior diagram in UML (Unified Modeling or the behavior of a system by illustrating the sequence of actions, decisions, and control business process. Activity diagrams are used to flow. The key elements and concepts in an activity diagram:

1. Activity: An activity represents a specific task, action, or operation that occurs within the system or process being modeled. Activities are usually depicted as rounded rectangles with a label describing the action.

2. Initial Node: An initial node,represented by a small filled circle,signifies the starting point of the activity diagram.It indicates where the execution of the activities begins.

3. Final Node : An final node, represented by a circle with a dot inside or the word "final", indicates the end of the activities or process. It marks the point at which the activities terminate.

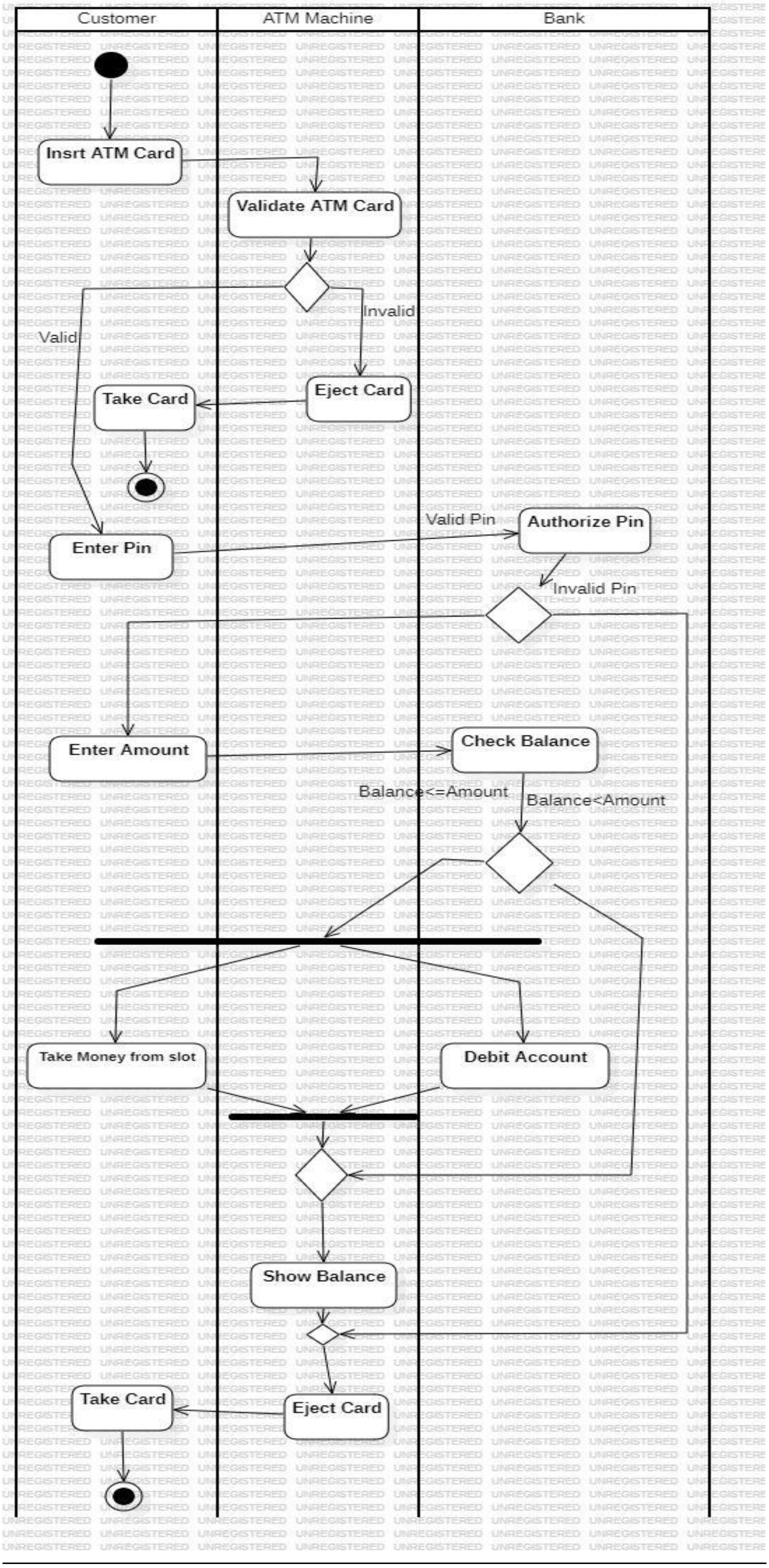
4. Decision Node : A decision node, represented as a diamond shape, is used to represent a decision point or a branching point in the process. It is associated with conditions or guards that determine the path the flow will take based on the evaluation of these conditions.

5. Merge Node : A merge node, also depicted as a diamond shape, is used to merge multiple control flows back into one. It signifies the point at which multiple branches of the process converge.

6. Fork Node : A fork node, represented as a thick horizontal bar, is used to split a single control flow into multiple parallel flows. It signifies the start of concurrent or parallel activities.

7. Join Node : A join node, depicted as a thick horizontal bar, is used to combine multiple parallel control flows into a single flow. It signifies the point at which concurrent activities rejoin.

8. Swimlane : Swimlanes are used to partition the activities in an activity diagram, typically representing different organizational units, roles, or entities responsible for executing the activities. Each swimlane is a vertical or horizontal container with a label identifying the partition.


Customer
ATM Machine
Bank
Insrt ATM Card
Validate ATM Card
Invalid
Valid
Take Card
Eject Card
Valid Pin
Authorize Pin
Enter Pin
Invalid Pin
Enter Amount
Check Balance
Balance<=Amount
Balance<Amount
Take Money from slot
Debit Account
Show Balance
Take Card
Eject Card

Practical 9

Study and implementation of Component Diagrams.

A component diagram is a type of structural diagram in the Unified Modeling LAGUAGE (UML) that visualizes the high-level structure of a system or software application by representing its components, their relationships, and the interfaces they a system or software expose. Component diagrams are used to illustrate the organization of system components, their interactions, and the dependencies between them. The key elements and concepts in a component diagram:

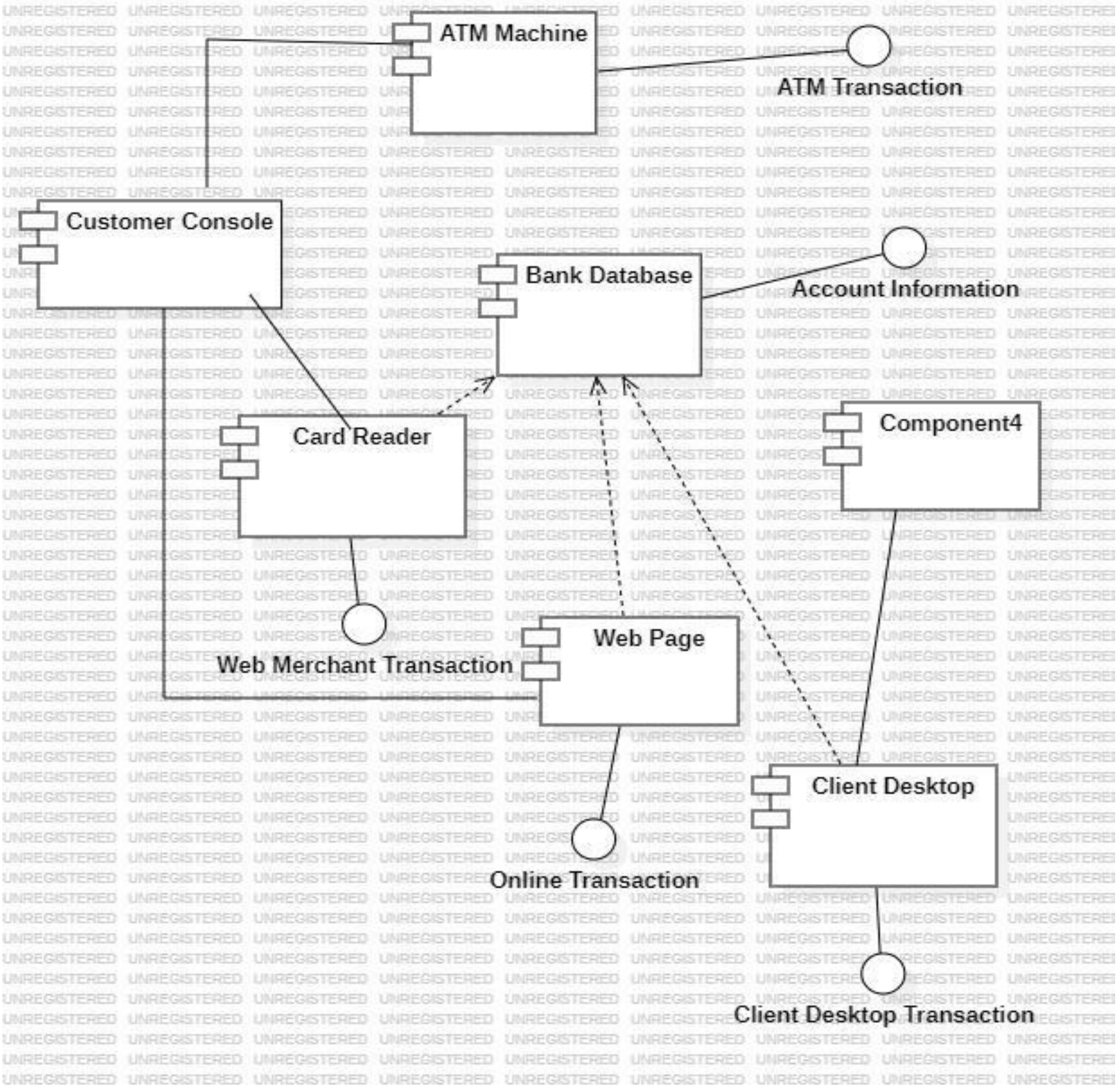
1.Component: A component represents a modular, reusable, and replaceable part or building block within a system. Components can be physical (e.g., hardware devices) or software-based (e.g., software modules, classes, libraries). They are typically depicted as rectangles with the component's name inside.

2. Interface: An interface defines a contract specifying the operations (methods or services) that a component provides or requires from other components. Interfaces are represented as a circle at the top of a component with the interface name, often preceded by "<<interface>>."

3. Dependency: A dependency relationship indicates that one component depends on another component. It represents a relationship in which changes in the source component may affect or require changes in the target component. Dependencies are shown as dashed arrows pointing from the dependent component to the target component.

4. Association: An association relationship represents a more general and less specific form of relationship between components. It is used to indicate that two components are somehow related or associated but do not necessarily imply a dependency. Associations are depicted as solid lines connecting the components.

5. Package : Components can be grouped within packages for organizational purposes. A package is a container that holds related components, and it is depicted as a folder-like icon. It helps manage and structure a system's components hierarchically.



Practical 10

Study and implementation of Deployment Diagrams.

A deployment diagram in UML (Unified Modeling Language) is a type of structural diagram that visualizes the physical deployment of software components (such as servers, hardware devices, and software nodes) within a system or an application.

Deployment diagrams help depict how software and hardware elements are distributed across different physical nodes or computing resources. They are commonly used in system architecture and design to ensure that a system can be effectively deployed and operated in a real-world environment. The key elements and concepts in a deployment diagram:

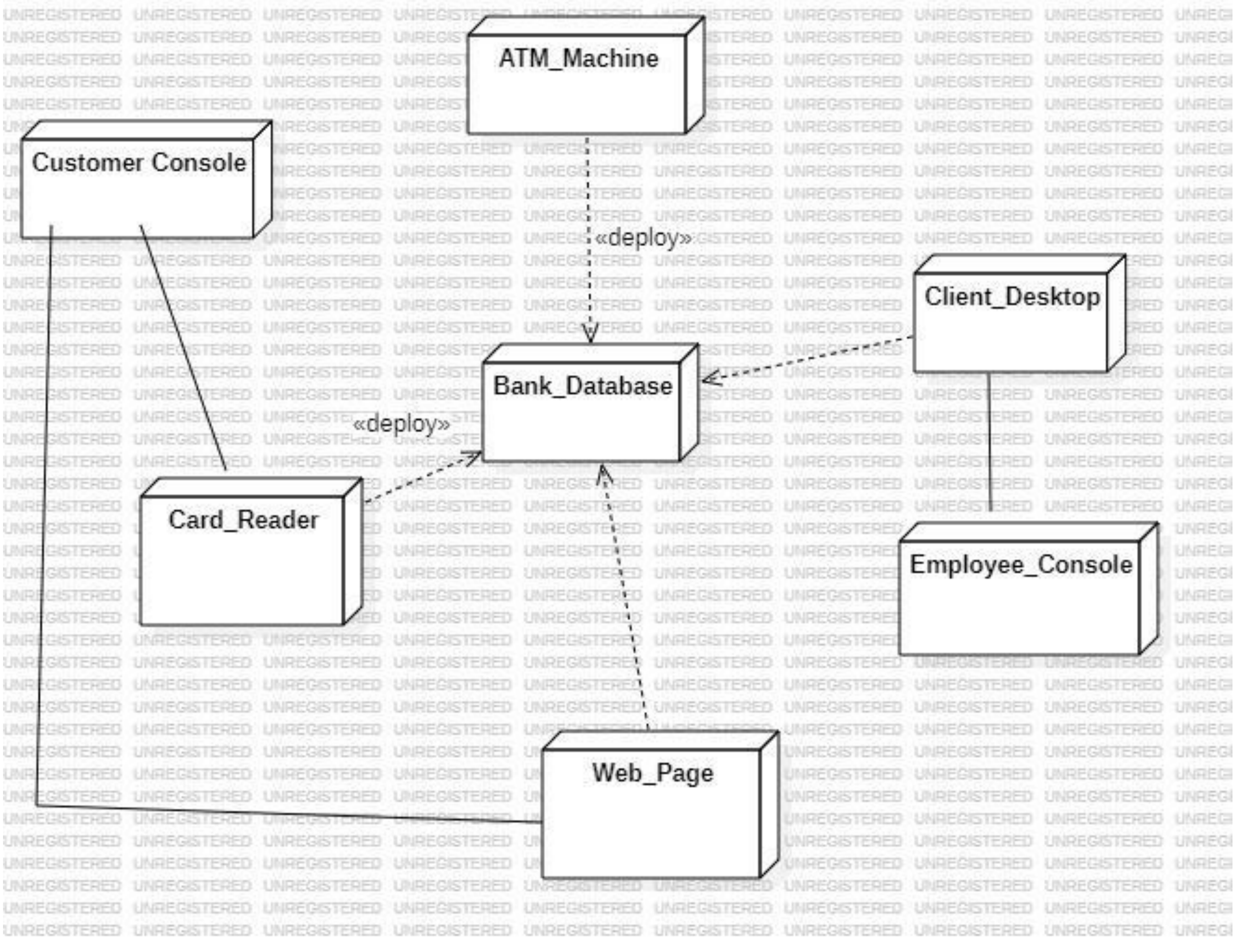
1. Node: A node represents a physical device or a software execution environment that hosts one or more components. Nodes can be servers, workstations, PCs, hardware devices, or even software containers like a Java Virtual Machine (JVM). Nodes are depicted as rectangles with the node's name inside.

2. Component: A component represents a software module, application, or executable unit that runs on a node. Components can include web applications, databases, services, or any software artifact. They are depicted as rectangles with the component's name inside.

3. Association: An association is used to depict the relationships or associations between nodes, components, and deployment artifacts. It shows how components are deployed on nodes and how nodes interact with each other. Associations are typically represented as solid lines connecting the elements.

4. Deployment Dependency: A deployment dependency is a relationship that represents a dependency between a component or artifact and a node. It indicates that a component or artifact relies on the presence or availability of a node to function correctly. It is typically shown as a dashed line with a "<<deploy>>" stereotype.

5. Communication Path: A communication path represents the network or communication channel between nodes. It illustrates how nodes communicate with each other or with external entities, such as clients or other systems. Communication paths are depicted as lines connecting nodes, often with labels indicating the type of communication (e.g., Ethernet, HTTP, TCP/IP).



Practical 3

Study and implementation of E R Diagrams.

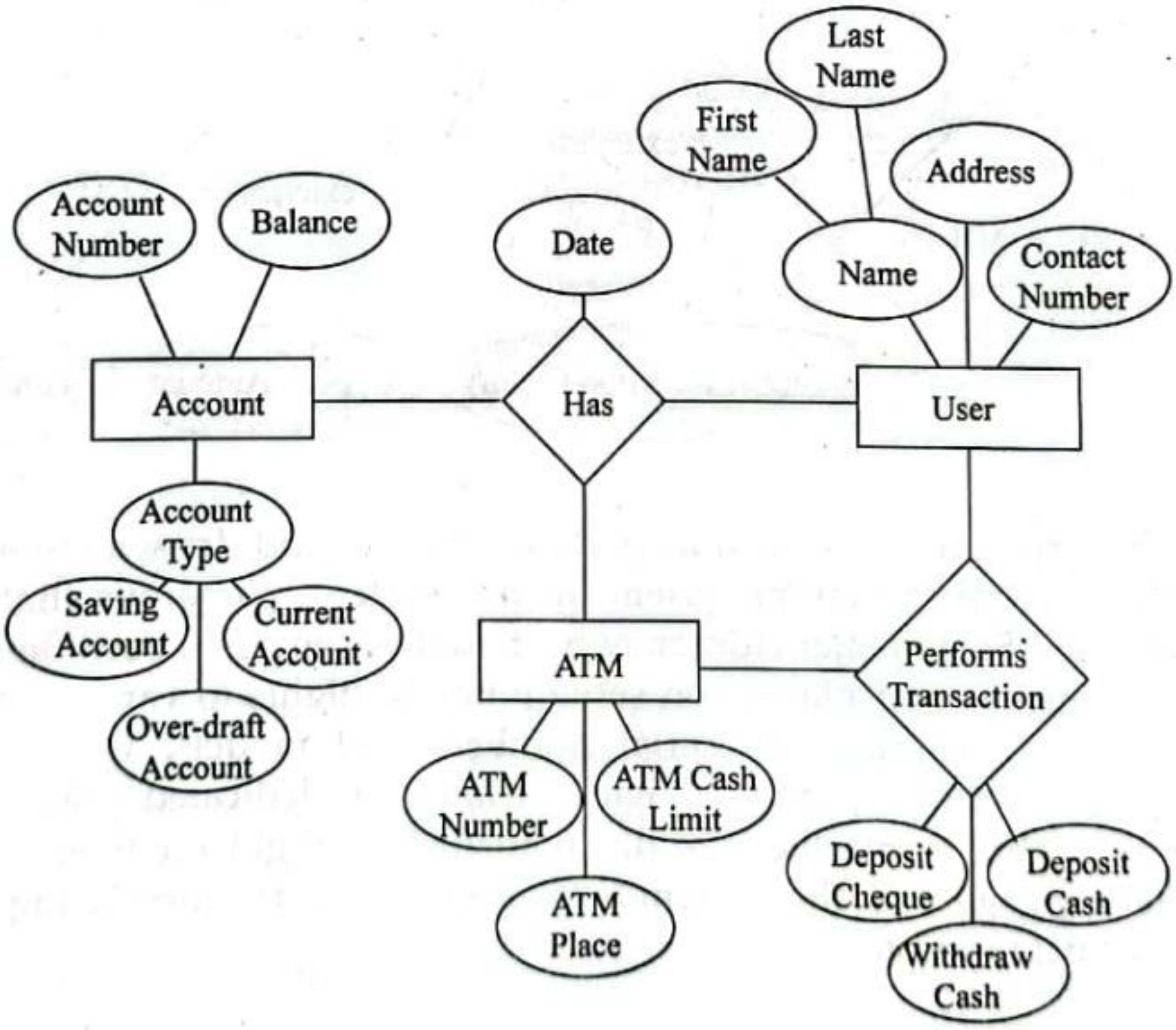
An Entity-Relationship (ER) diagram is a visual representation of the data model for a database system. ER diagrams are widely used in database design and modeling to depict the structure of a database, including the entities (objects or concepts), their attributes (properties), and the relationships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



-X-X-X-X-X-