SOFTWARE ENGINEERING

UNIT – 3

CHAPTER – 3

UML DESIGN

I. A conceptual model of the UML

A conceptual model in Unified Modeling Language (UML) refers to the high-level representation of a system's structure and behavior. It provides a simplified and abstract view of the system, helping stakeholders understand its key components and interactions.

Unified Modeling Language (UML) is a standardized modeling language used in software engineering for visualizing, specifying, constructing, and documenting the artifacts of a system. The basic building blocks of UML are the elements and diagrams that help model different aspects of a system.

Vocabulary of the UML encompasses three kinds of building blocks:

1. Things
2. Relationships
3. Diagrams

Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group interesting collections of things.

In Unified Modeling Language (UML), "things" refer to the various elements and constructs used to model the structure and behavior of a system. There are four kinds of things in the UML:

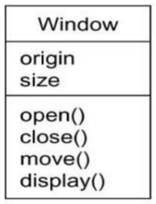
1. Structural things
2. Behavioral things
3. Grouping things
4. Annotational things

These things are the basic object-oriented building blocks of the UML.

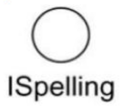
1. Structural Things:

Structural elements in UML represent the static aspects of a system, describing the composition, relationships, and organization of its components. In all, there are seven kinds of structural things.

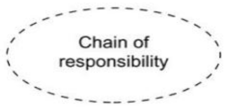
First, a **class** is a description of a set of objects that share the same attributes, operations, relationships, and semantics. A class implements one or more interfaces. Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations.



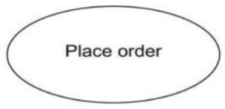
Second, an **interface** is a collection of operations that specify a service of a class or component. An interface therefore describes the externally visible behavior of that element. An interface might represent the complete behavior of a class or component or only a part of that behavior. Graphically, an interface is rendered as a circle together with its name. An interface rarely stands alone. Rather, it is typically attached to the class or component that realizes the interface.



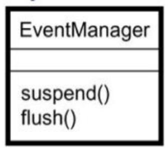
Third, a **collaboration** defines an interaction and is a society of roles and other elements that work together to provide some cooperative behavior. Graphically, a collaboration is rendered as an ellipse with dashed lines, usually including only its name.



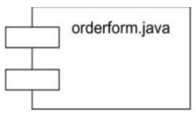
Fourth, a **use case** is a representation of a specific interaction between a system (or its components) and an external actor, which can be a user, another system, or any external entity. Use cases help to describe the functional requirements of a system from an external user's perspective. Graphically, a use case is rendered as an ellipse with solid lines, usually including only its name.



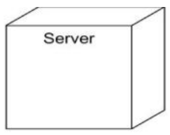
Fifth, an **active class** is a class that plays a special role in modeling concurrent or parallel behavior within a system. An active class is used to represent an object that can execute operations concurrently with other objects. The concept of active classes is particularly relevant in the context of concurrent or parallel programming, where multiple tasks or processes can be executed simultaneously. Graphically, an active class is rendered just like a class, but with heavy lines, usually including its name, attributes, and operations.



Sixth, a **component** is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces. A component is a modular and deployable unit of software that encapsulates its contents and provides a well-defined interface. Components are used to represent and model the physical or logical building blocks of a system. They can be libraries, executables, documents, or any other artifacts that are part of the system. Graphically, a component is rendered as a rectangle with tabs, usually including only its name, as



Seventh, a **node** is a physical element that exists at run time and represents a computational resource, generally having at least some memory and, often, processing capability. A set of components may reside on a node and may also migrate from node to node. Graphically, a node is rendered as a cube, usually including only its name.

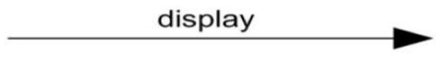


2. Behavioral Things:

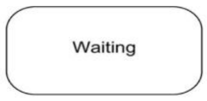
Behavioral things are the dynamic parts of UML models. These are the verbs of a model, representing behavior over time and space. In all, there are two primary kinds of behavioral things.

First, an interaction is a behavior that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose. An interaction involves a number of other elements, including messages, action sequences (the behavior invoked by a message), and links (the connection between objects).

Graphically, a message is rendered as a directed line, almost always including the name of its operation.



Second, a state machine is a behavior that specifies the sequences of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events. The behavior of an individual class or a collaboration of classes may be specified with a state machine. A state machine involves a number of other elements, including states, transitions (the flow from state to state), events (things that trigger a transition), and activities (the response to a transition). Graphically, a state is rendered as a rounded rectangle, usually including its name and its sub states.

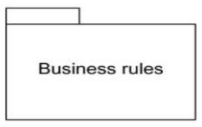


These two elements• interactions and state machines• are the basic behavioral things that you may include in a UML model. Semantically, these elements are usually connected to various structural elements, primarily classes, collaborations, and objects.

3. Grouping Things

Grouping things are the organizational parts of UML models. These are the boxes into which a model can be decomposed. In all, there is one primary kind of grouping thing, namely, packages.

A package is a general-purpose mechanism for organizing elements into groups. Structural things, behavioral things, and even other grouping things may be placed in a package. Unlike components (which exist at run time), a package is purely conceptual (meaning that it exists only at development time). Graphically, a package is rendered as a tabbed folder, usually including only its name and, sometimes, its contents.

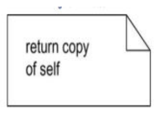


Packages are the basic grouping things with which you may organize a UML model. There are also variations, such as frameworks, models, and subsystems (kinds of packages).

4. **Annotational Things**

Annotations provide a way to add supplementary information or comments to a model without affecting its semantics. Annotations are not formal UML elements but are commonly used to enhance the clarity and understanding of a model. There is one primary kind of annotational thing, called a note.

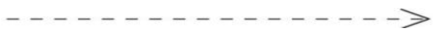
A note is simply a symbol for rendering constraints and comments attached to an element or a collection of elements. Graphically, a note is rendered as a rectangle with a dog-eared corner, together with a textual or graphical comment.



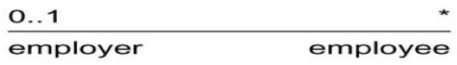
RELATIONSHIPS IN THE UML

Unified Modeling Language (UML) provides several types of relationships to model the associations and connections between different elements in a system. These relationships help to define the structure, behavior, and interactions within a system. There are four kinds of relationships in the UML:

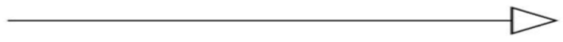
First, a **dependency** Represents a relationship where a change in one element may affect another. Indicates that a client element depends on a supplier element. Denoted by a dashed arrow pointing from the dependent to the independent element.



Second, an **association** Represents a bi-directional relationship between two classes. Indicates a structural connection between instances of the associated classes. Can be annotated with multiplicity to specify the number of instances involved.



Third, a ***generalization*** Represents an "is-a" relationship between a general (superclass) and a specific (subclass) class. Allows the subclass to inherit attributes and behaviors from the superclass. Denoted by a solid line with an open arrowhead pointing to the superclass.



Fourth, a ***realization,*** represents the implementation of an interface by a class or a component. Indicates that the realizing class provides the operations defined by the interface. Denoted by a dashed line with an open arrowhead pointing to the interface.

DIAGRAMS IN THE UML

Unified Modeling Language (UML) diagrams are widely used in software engineering and other fields for several reasons:

a) Visualization: UML diagrams provide a visual representation of complex systems, making it easier for stakeholders to understand and comprehend the system's structure, behavior, and interactions.

b) Communication: UML diagrams serve as a common language for communication among stakeholders, including developers, analysts, designers, and clients. They offer a visual way to convey ideas and concepts.

c) Specification: UML allows for the specification of system requirements, design, and architecture in a standardized and unambiguous manner. This helps ensure that all parties involved have a shared understanding of the system.

d) Design and Analysis: UML supports various types of diagrams, each focusing on specific aspects of a system. This makes it a versatile tool for both high-level system design and detailed analysis of specific components or behaviors.

e) Documentation: UML diagrams serve as documentation for software projects, providing a visual record of decisions made during the design and development process. This documentation is valuable for future maintenance and updates.

f) Analysis and Modeling: UML supports the modeling of system requirements, use cases, classes, objects, interactions, and more. This modeling capability helps analysts and designers explore and refine system specifications.

g) Code Generation: Some UML tools support code generation, allowing developers to automatically generate code from UML diagrams. This helps maintain consistency between the design and implementation phases.

h) System Understanding: UML diagrams aid in gaining a holistic understanding of a system by capturing its various aspects, including static structure, dynamic behavior, and deployment.

i) Iterative Development: UML supports an iterative and incremental development process. As a project evolves, UML diagrams can be updated and refined to reflect changes in requirements, design decisions, or implementation details.

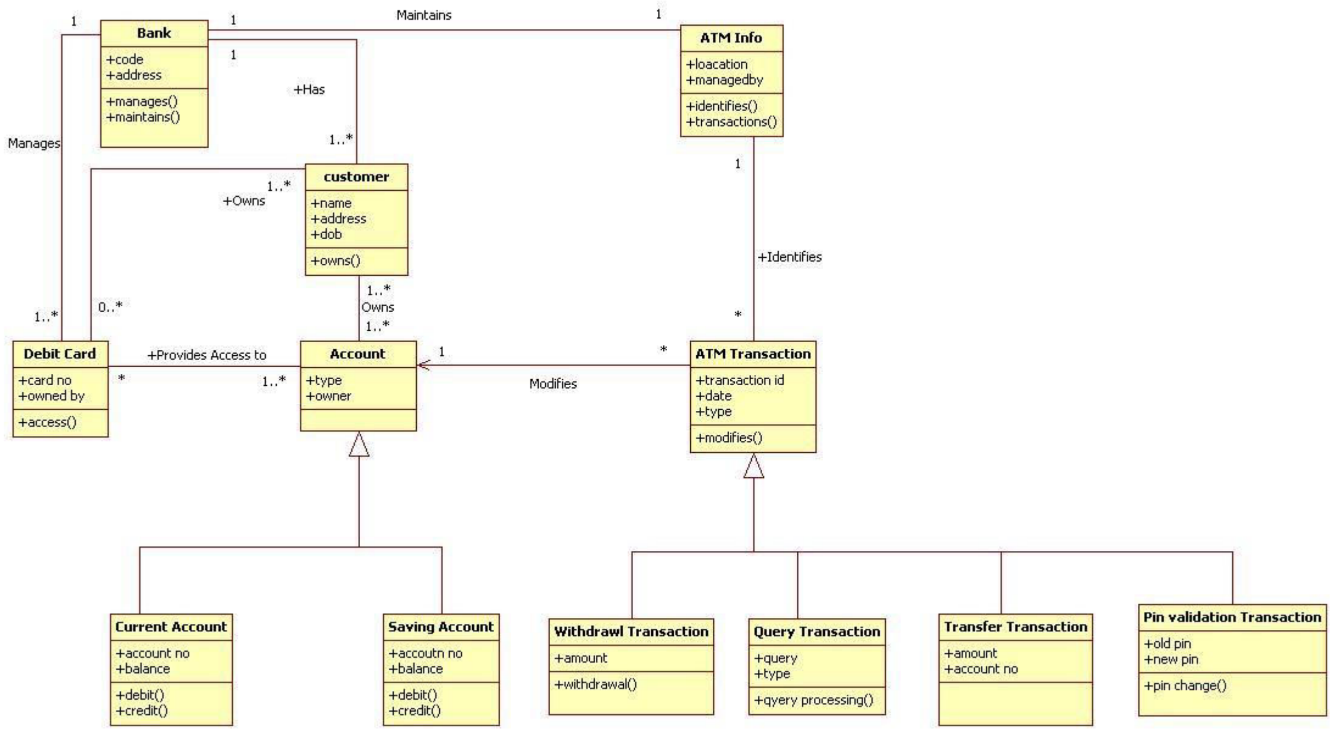
j) Collaboration: UML diagrams facilitate collaboration among team members, enabling them to discuss, review, and refine the system design collaboratively. This is particularly useful in distributed or interdisciplinary development environments.

k) Problem Solving: UML diagrams help in problem-solving by providing a clear and visual representation of the system. They allow teams to identify potential issues, refine designs, and make informed decisions. For this reason, the UML includes nine such diagrams:

1. Class diagram
2. Object diagram
3. Use case diagram
4. Sequence diagram
5. Collaboration diagram
6. State chart diagram
7. Activity diagram
8. Component diagram
9. Deployment diagram

Class diagrams:

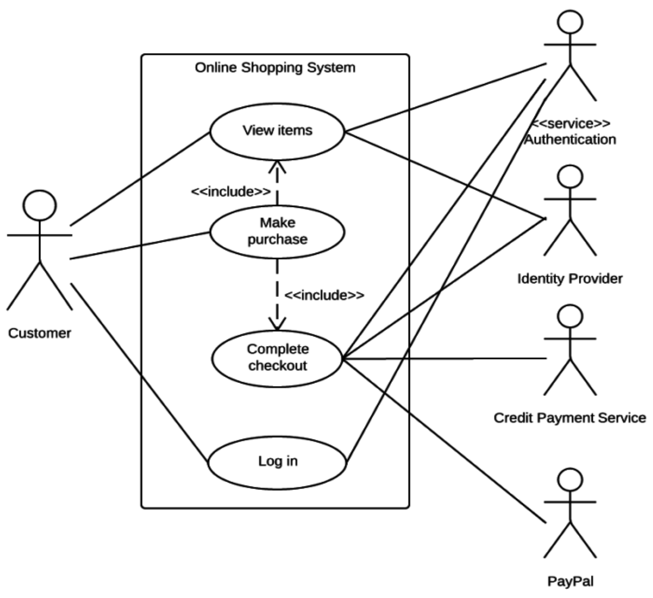
Illustrates the static structure of a system, showing classes, their attributes, methods, and relationships. Used for modeling the object-oriented aspects of a system.



Class diagram for ATM system

Use case diagrams:

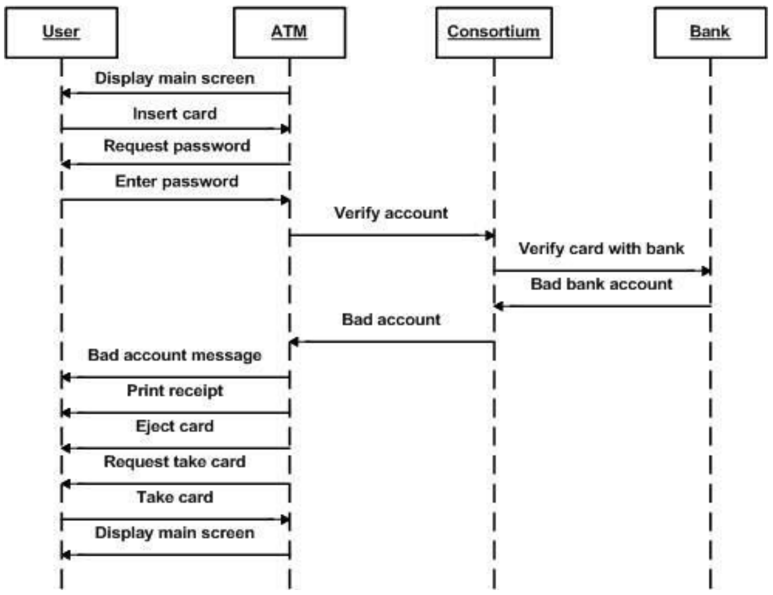
Describes the functional requirements of a system from an external user's perspective. Represents use cases, actors, and their relationships to model system behavior.



Use case diagram for Online Shopping system

Interaction diagrams – Sequence Diagram:

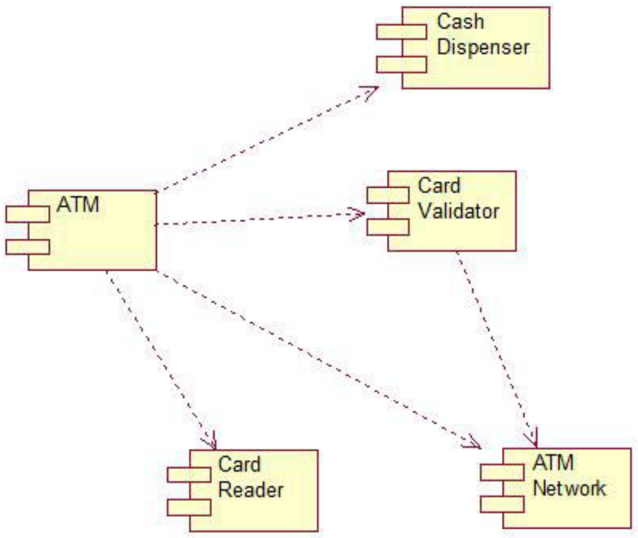
Depicts the interactions between objects or components over time. Shows the order of messages exchanged between objects in response to various stimuli.



Sequence diagram for ATM system.

Component diagrams:

Depicts the physical components of a system and their dependencies. Illustrates the organization and dependencies among components in the implementation of a system.



Component diagram for ATM system