Class Diagram:

The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.

It shows the attributes, classes, functions, and relationships to give an overview of the software system. It constitutes class names, attributes, and functions in a separate compartment that helps in software development. Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram.

Purpose of Class Diagrams

The main purpose of class diagrams is to build a static view of an application. It is the only diagram that is widely used for construction, and it can be mapped with object-oriented languages. It is one of the most popular UML diagrams. Following are the purpose of class diagrams given below:

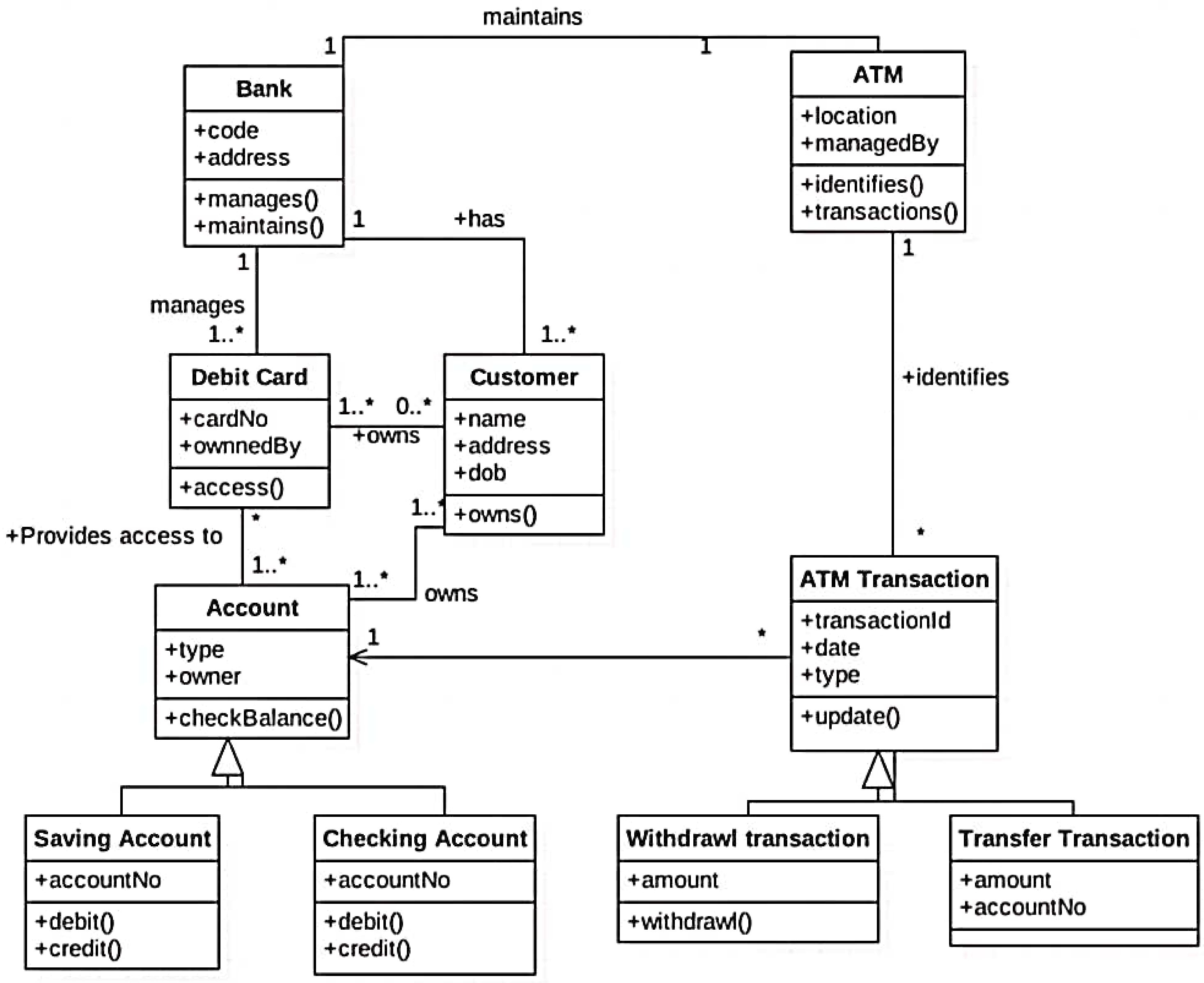
1. It analyses and designs a static view of an application.
2. It describes the major responsibilities of a system.
3. It is a base for component and deployment diagrams.
4. It incorporates forward and reverse engineering.

Benefits of Class Diagrams

1. It can represent the object model for complex systems.
2. It reduces the maintenance time by providing an overview of how an application is structured before coding.
3. It provides a general schematic of an application for better understanding.
4. It represents a detailed chart by highlighting the desired code, which is to be programmed.
5. It is helpful for the stakeholders and the developers.

1. Class Diagram for ATM

This class diagram for the atm maps out the structure and attributes of how an ATM works. It also shows the relationship between multiple classes. You can use this template as it is or modify it according to your needs.



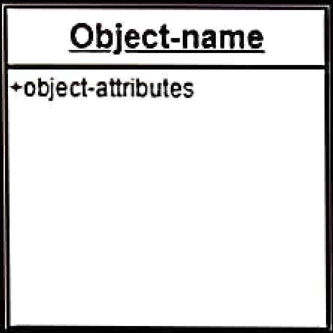
4. To construct the software application using object-oriented languages.

Object Diagram

Object diagrams are dependent on the class diagram as they are derived from the class diagram. It represents an instance of a class diagram. The objects help in portraying a static view of an object-oriented system at a specific instant.

Both the object and class diagram are similar to some extent; the only difference is that the class diagram provides an abstract view of a system. It helps in visualizing a particular functionality of a system.

Notation of an Object Diagram



Purpose of Object Diagram

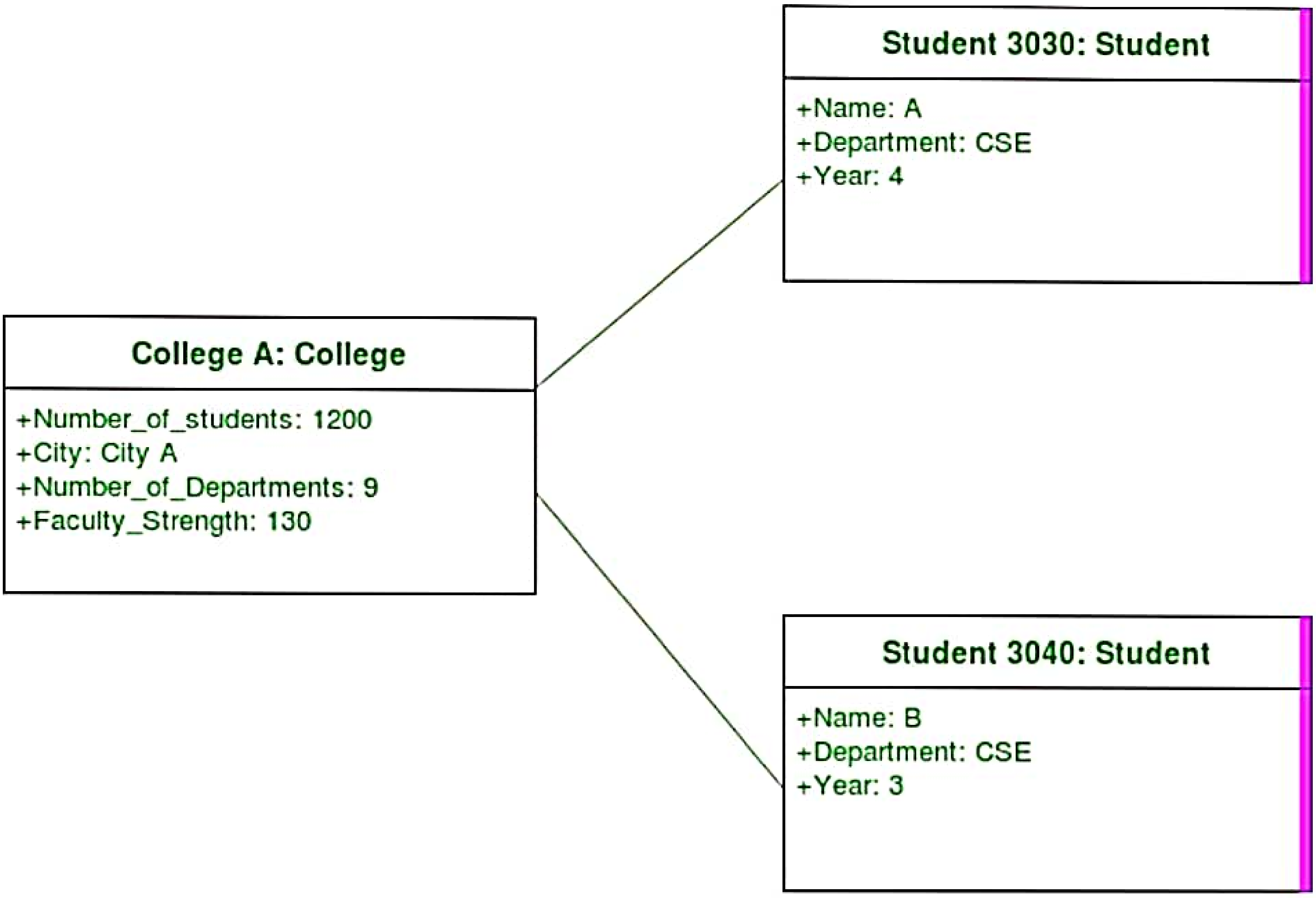
The object diagram holds the same purpose as that of a class diagram. The class diagram provides an abstract view which comprises of classes and their relationships, whereas the object diagram represents an instance at a particular point of time.

The object diagram is actually similar to the concrete (actual) system behavior. The main purpose is to depict a static view of a system.

Steps to draw an Object Diagram:

1. All the objects present in the system should be examined before start drawing the object diagram.
2. Before creating the object diagram, the relation between the objects must be acknowledged.
3. The association relationship among the entities must be cleared already.
4. To represent the functionality of an object, a proper meaningful name should be assigned.
5. The objects are to be examined to understand its functionality.

For example – In the figure below, two objects of class Student are linked to an object of class College.



Component Diagram:

The component diagrams have remarkable importance. It is used to depict the functionality and behavior of all the components present in the system, unlike other diagrams that are used to represent the architecture of the system, working of a system, or simply the system itself.

In UML, the component diagram portrays the behavior and organization of components at any instant of time. The system cannot be visualized by any individual component, but it can be by the collection of components.

Following are some reasons for the requirement of the component diagram:

1. It portrays the components of a system at the runtime.
2. It is helpful in testing a system.
3. It envisions the links between several connections.

It represents various physical components of a system at runtime. It is helpful in visualizing the structure and the organization of a system. It describes how individual components can together form a single system. Following are some reasons, which tells when to use component diagram:

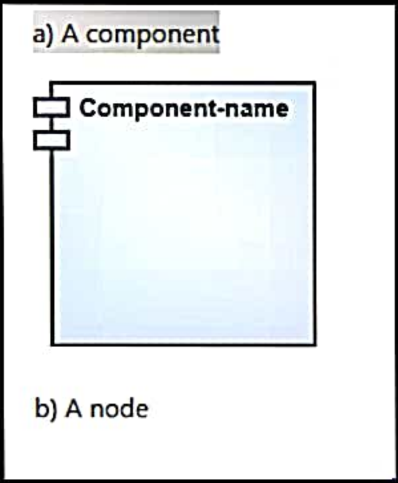
1. To divide a single system into multiple components according to the functionality.
2. To represent the component organization of the system.

The component diagram is helpful in representing the physical aspects of a system, which are files, executables, libraries, etc. The main purpose of a component diagram is different from that of other diagrams. It is utilized in the implementation phase of any application.

Some of the artifacts that are needed to be identified before drawing a component diagram:

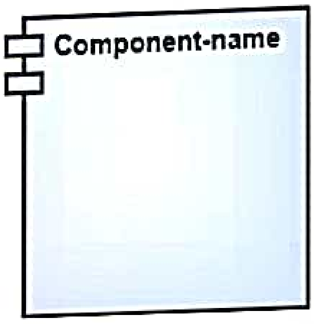
1. What files are used inside the system?
2. What is the application of relevant libraries and artifacts?
3. What is the relationship between the artifacts?

Notation of a Component Diagram

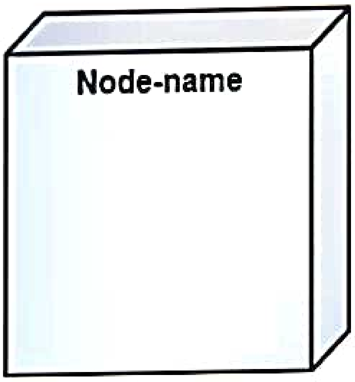


Notation of a Component Diagram

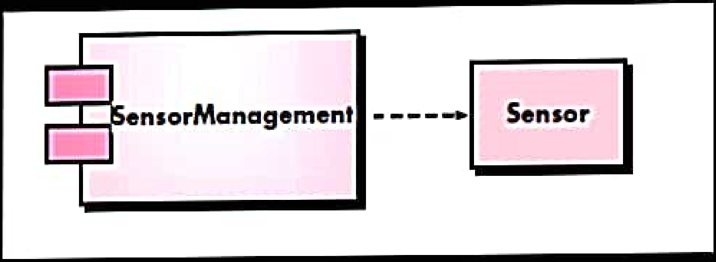
a) A component



b) A node

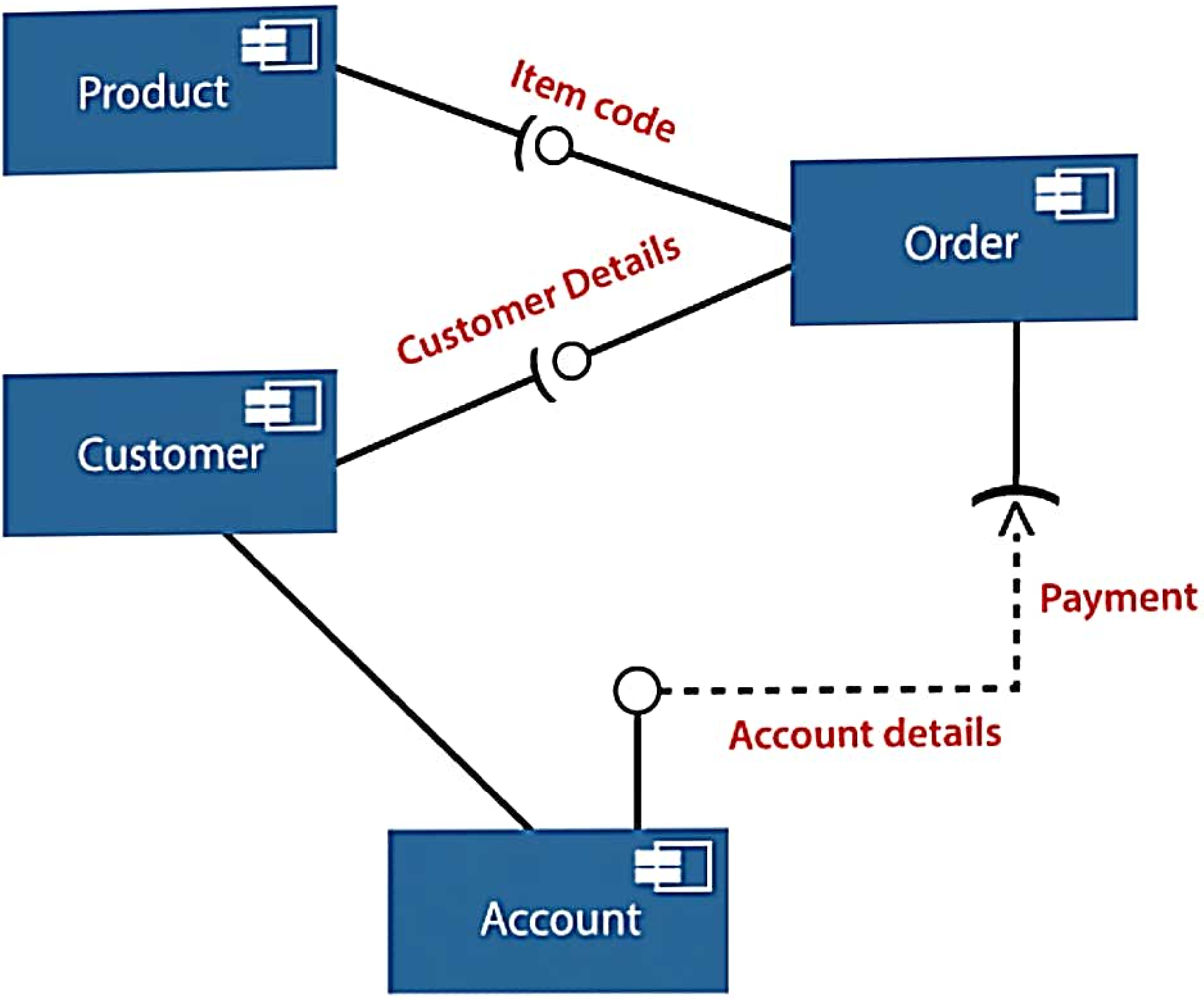


Ex:



In this figure, a component named SensorManagement (part of the *SafeHome* security function) is represented. A dashed arrow connects the component to a class named Sensor that is assigned to it. The SensorManagement component performs all functions associated with *SafeHome* sensors including monitoring and configuring them.

A component diagram for an online shopping system is given below:



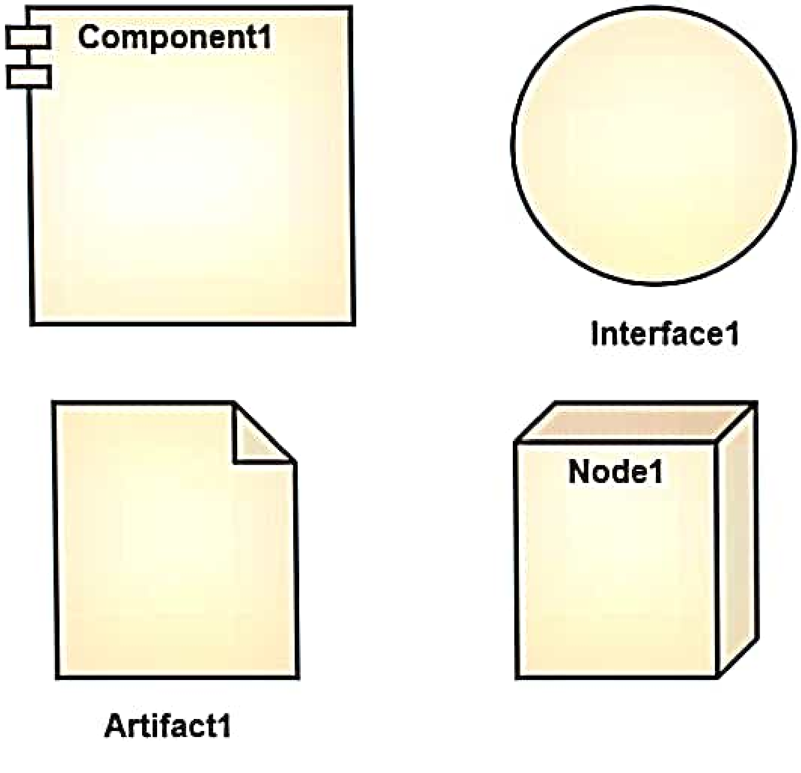
Deployment diagram:

The main purpose of the deployment diagram is to represent how software is installed on the hardware component. It depicts in what manner a software interacts with hardware to perform its execution.

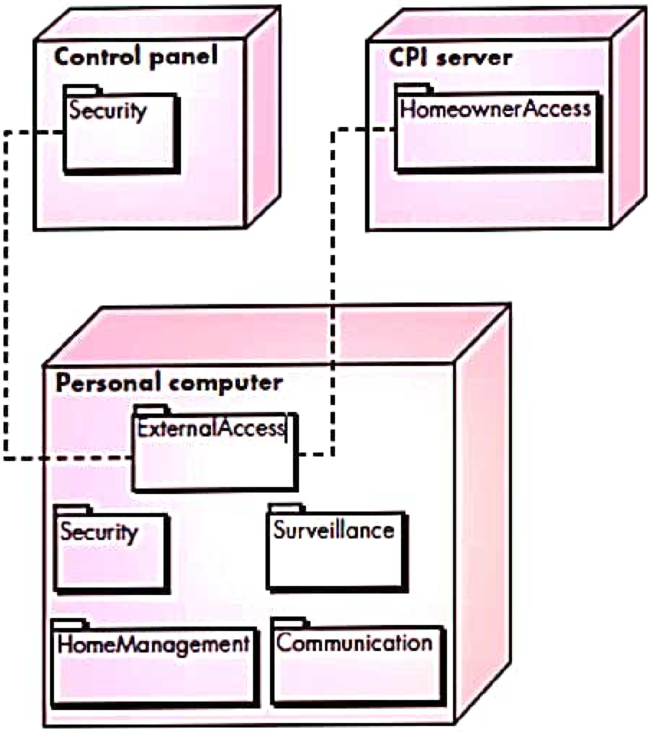
Both the deployment diagram and the component diagram are closely interrelated to each other as they focus on software and hardware components. The component diagram represents the components of a system, whereas the deployment diagram describes how they are actually deployed on the hardware.

The deployment diagram consist of the following notations:

1. A component
2. An artifact
3. An interface
4. A node



Ex:



In the figure, three computing environments are shown (in actuality, there would be more including sensors, cameras, and others). The subsystems (functionality) housed within each computing element are indicated. For example, the personal computer houses subsystems that implement security, surveillance, home management, and communications features. In addition, an external access subsystem has been designed to manage all attempts to access the *SafeHome* system from an external source. Each subsystem would be elaborated to indicate the components that it implements.

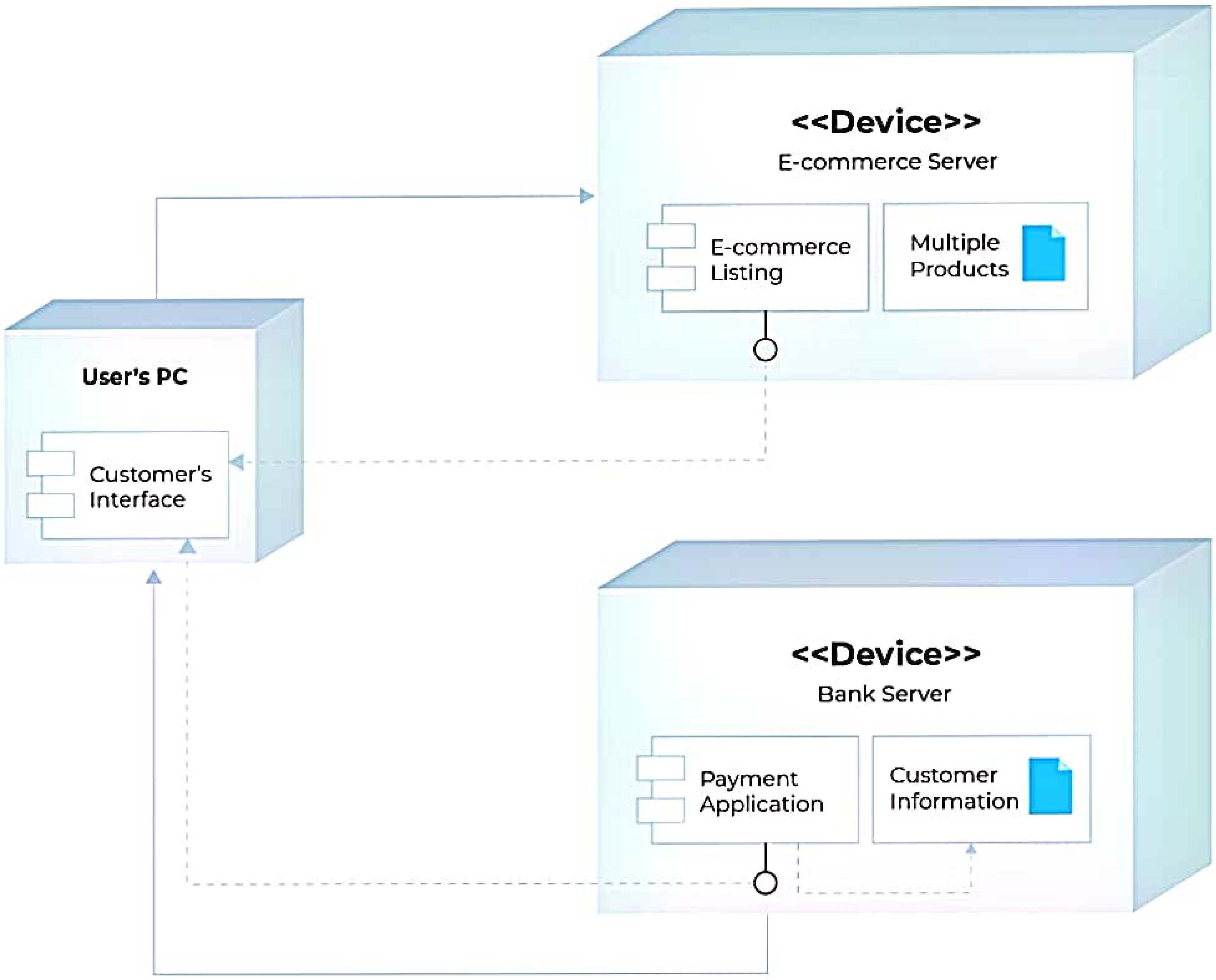
The diagram shown in Figure 8.7 is in *descriptor form*. This means that the deployment diagram shows the computing environment but does not explicitly indicate configuration details. For example, the "personal computer" is not further identified.

It could be a Mac or a Windows-based PC, a Sun workstation, or a Linux-box. These details are provided when the deployment diagram is revisited in *instance form* during the latter stages of design or as construction begins. Each instance of the deployment (a specific, named hardware configuration) is identified.

Deployment diagrams can be used for the followings:

1. To model the network and hardware topology of a system.
2. To model the distributed networks and systems.
3. Implement forwarding and reverse engineering processes.
4. To model the hardware details for a client/server system.
5. For modeling the embedded system.

Example 2: Deployment Diagram for Online Payment Services



This example shows how an e-commerce platform receives payment from a customer for a product. There's an

Behavioural Diagrams:

Use case diagram:-

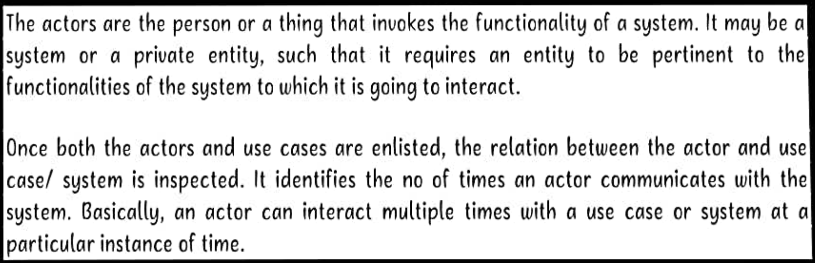
A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships.

It models the tasks, services, and functions required by a system/subsystem of an application.

It depicts the high-level functionality of a system and also tells how the user handles a system.

The main purpose of use case diagram is as follows.

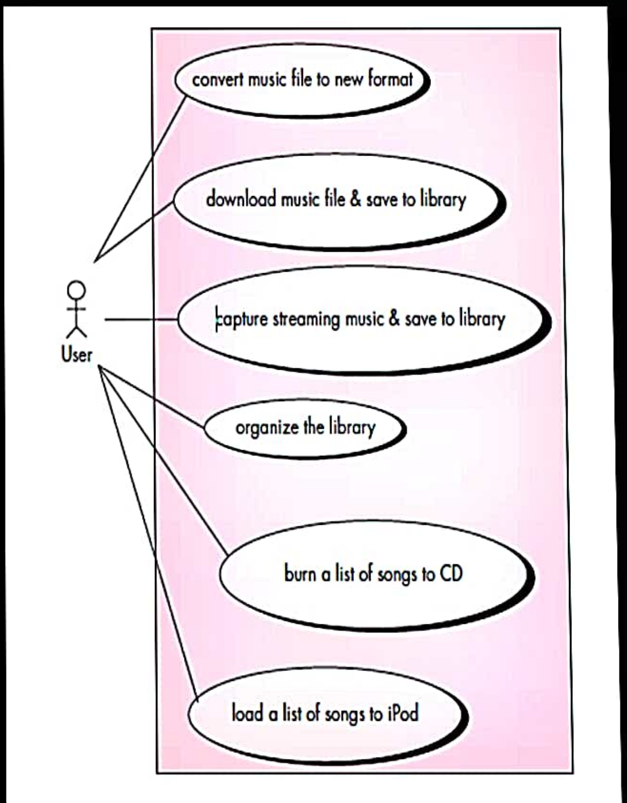
1. It gathers the system's needs.
2. It depicts the external view of the system.
3. It recognizes the internal as well as external factors that influence the system.
4. It represents the interaction between the actors.



The actors are the person or a thing that invokes the functionality of a system. It may be a system or a private entity, such that it requires an entity to be pertinent to the functionalities of the system to which it is going to interact.

Once both the actors and use cases are enlisted, the relation between the actor and use case/ system is inspected. It identifies the no of times an actor communicates with the system. Basically, an actor can interact multiple times with a use case or system at a particular instance of time.

Ex: Use case diagram for the music system



User
convert music file to new format
download music file & save to library
capture streaming music & save to library
organize the library
burn a list of songs to CD
load a list of songs to iPod

Sequence Diagram

The sequence diagram represents the flow of messages in the system and is also termed as an event diagram. It helps in envisioning several dynamic scenarios. It portrays the communication between any two lifelines as a time-ordered sequence of events, such that these lifelines took part at the run time. In UML, the lifeline is represented by a vertical bar, whereas the message flow is represented by a vertical dotted line that extends across the bottom of the page. It incorporates the iterations as well as branching.

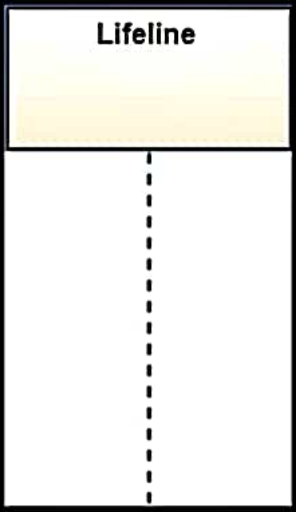
Purpose of a Sequence Diagram

1. To model high-level interaction among active objects within a system.
2. To model interaction among objects inside a collaboration realizing a use case.
3. It either models generic interactions or some certain instances of interaction.

Notations of a Sequence Diagram

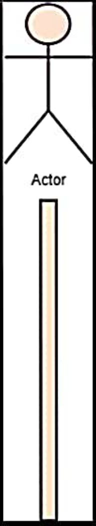
Lifeline

An individual participant in the sequence diagram is represented by a lifeline. It is positioned at the top of the diagram.



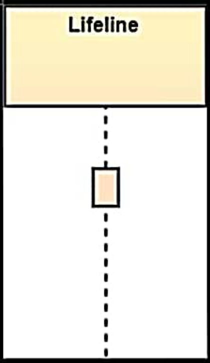
Actor

A role played by an entity that interacts with the subject is called as an actor. It is out of the scope of the system. It represents the role, which involves human users and external hardware or subjects. An actor may or may not represent a physical entity, but it purely depicts the role of an entity. Several distinct roles can be played by an actor or vice versa.



Activation

It is represented by a thin rectangle on the lifeline. It describes that time period in which an operation is performed by an element, such that the top and the bottom of the rectangle is associated with the initiation and the completion time, each respectively.

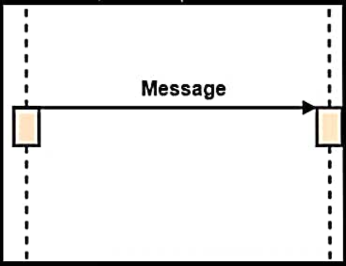


Messages

The messages depict the interaction between the objects and are represented by arrows. They are in the sequential order on the lifeline. The core of the sequence diagram is formed by messages and lifelines.

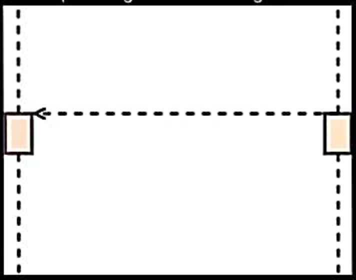
Following are types of messages enli

- **Call Message:** It defines a par tion between the lifelines of an interaction, which represents that the target lifeline has invoked an operation.

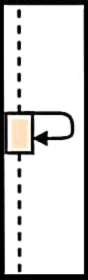


- **Return Message:** It defines a particular communication between the lifelines of

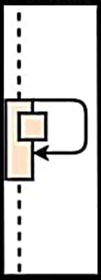
- **Return Message:** It defines a particular communication between the lifelines of interaction that represent the flow of information from the receiver of the corresponding caller message.



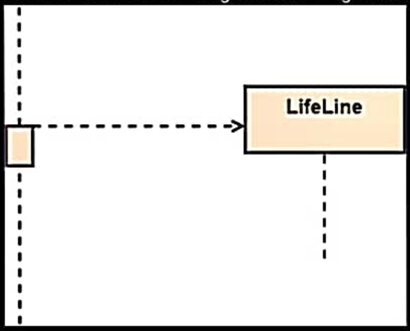
- **Self Message:** It describes a communication, particularly between the lifelines of an interaction that represents a message of the same lifeline, has been invoked.



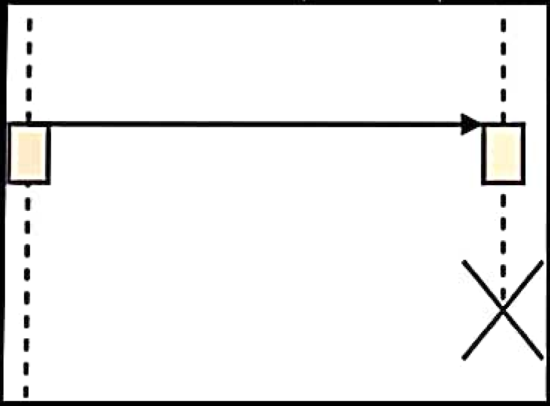
- **Recursive Message:** A self message sent for recursive purpose is called a recursive message. In other words, it can be said that the recursive message is a special case of the self message as it represents the recursive calls.



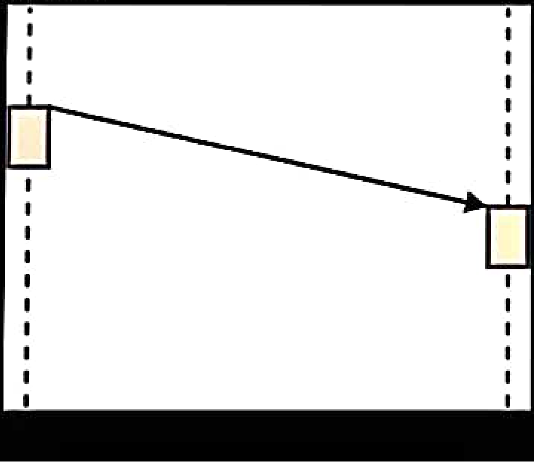
- **Create Message:** It describes a communication, particularly between the lifelines of an interaction describing that the target (lifeline) has been instantiated.



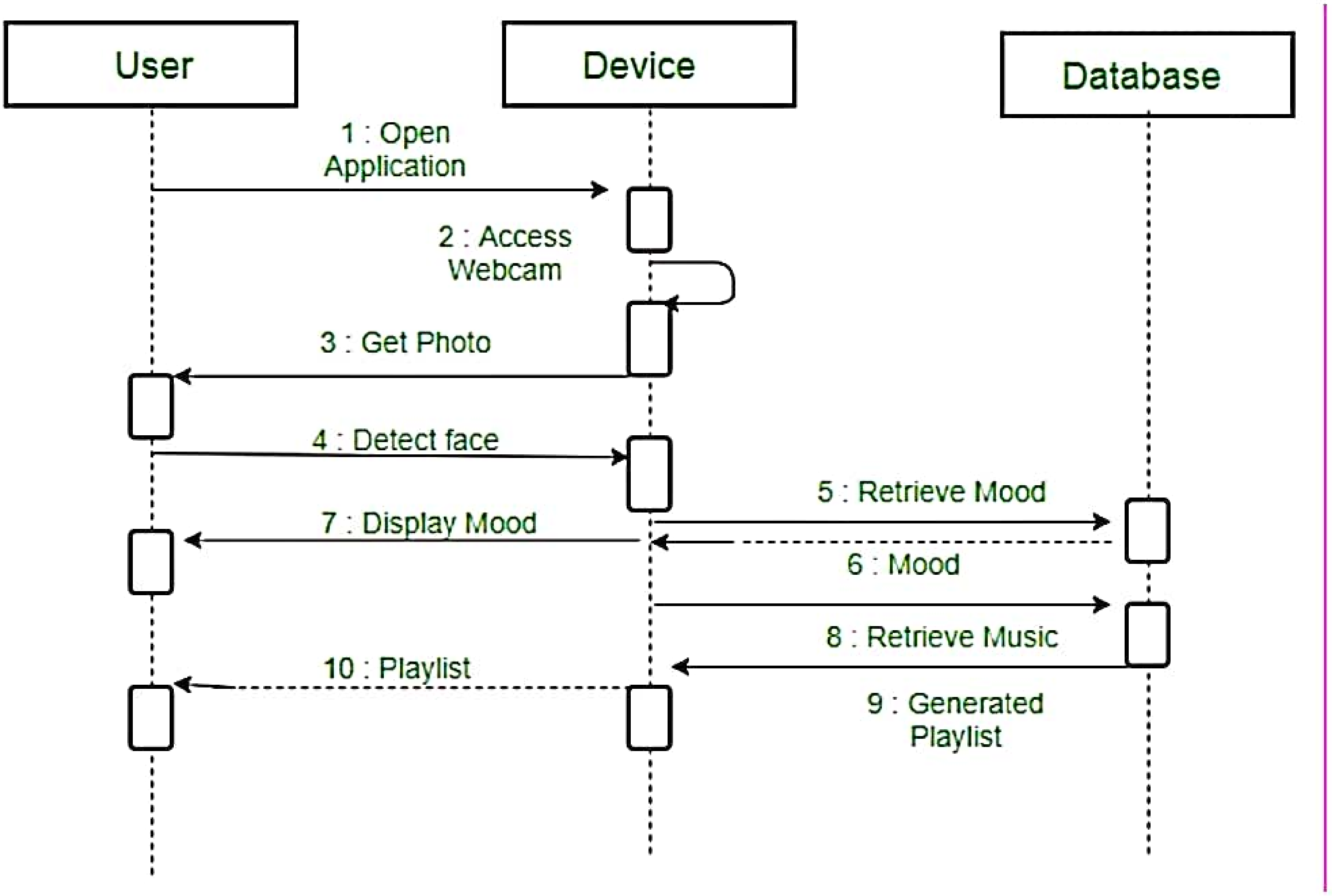
- **Destroy Message:** It describes a communication, particularly between the lifelines of an interaction that depicts a request to destroy the lifecycle of the target.



- **Duration Message:** It describes a communication particularly between the lifelines of an interaction, which portrays the time passage of the message while modeling a system.



A sequence diagram for an emotion based music player –



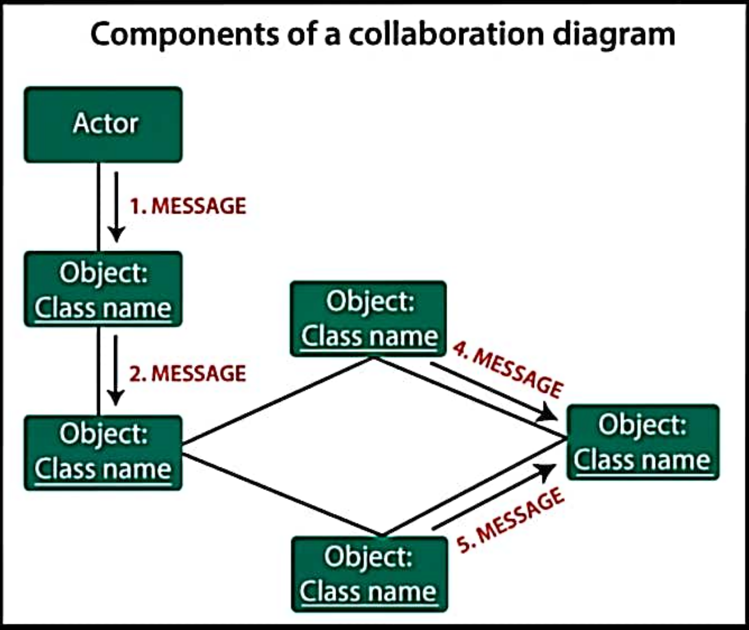
Collaboration diagram:

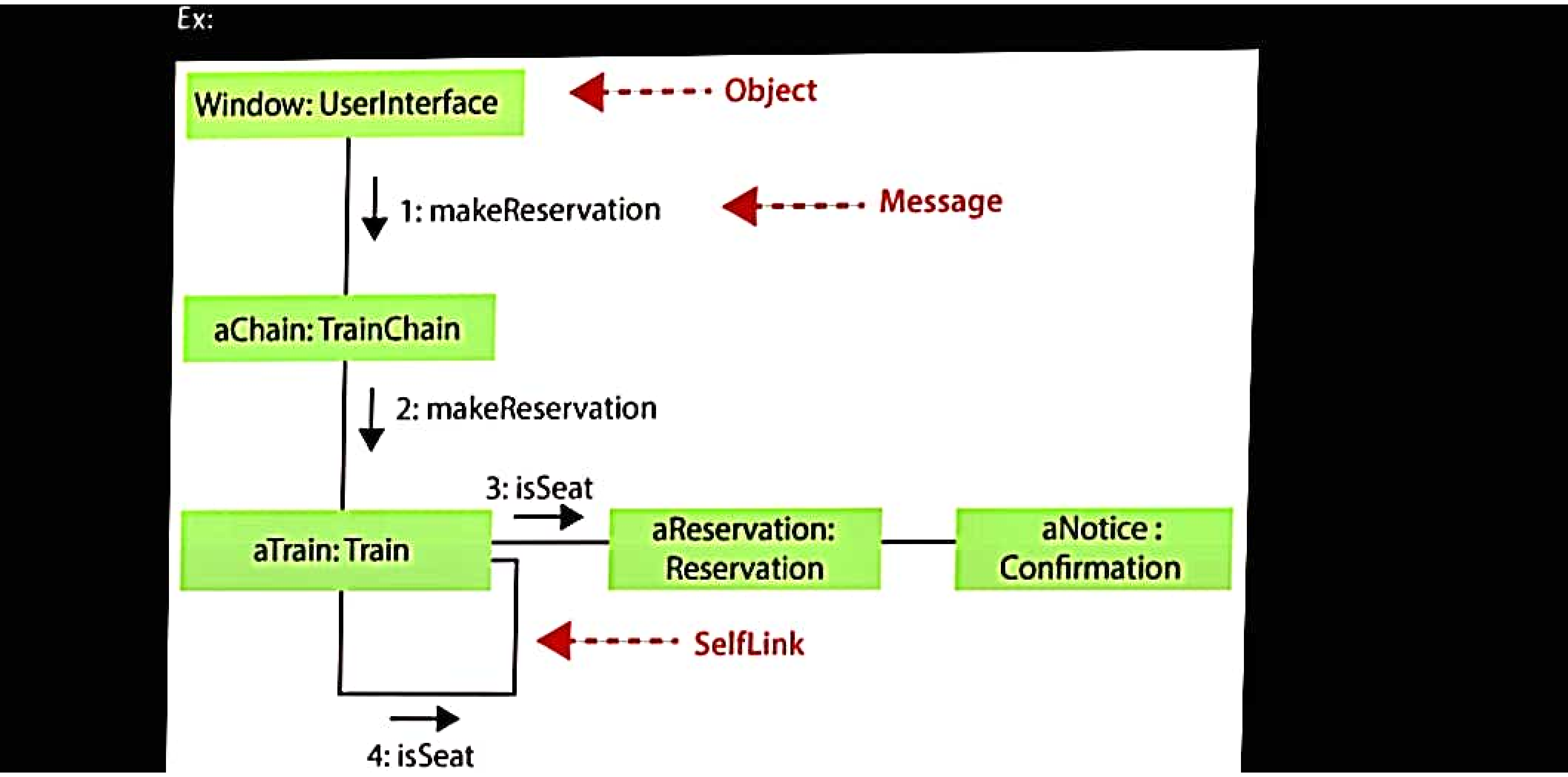
The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming. An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.

Notations of a Collaboration Diagram

Following are the components of a component diagram that are enlisted below:

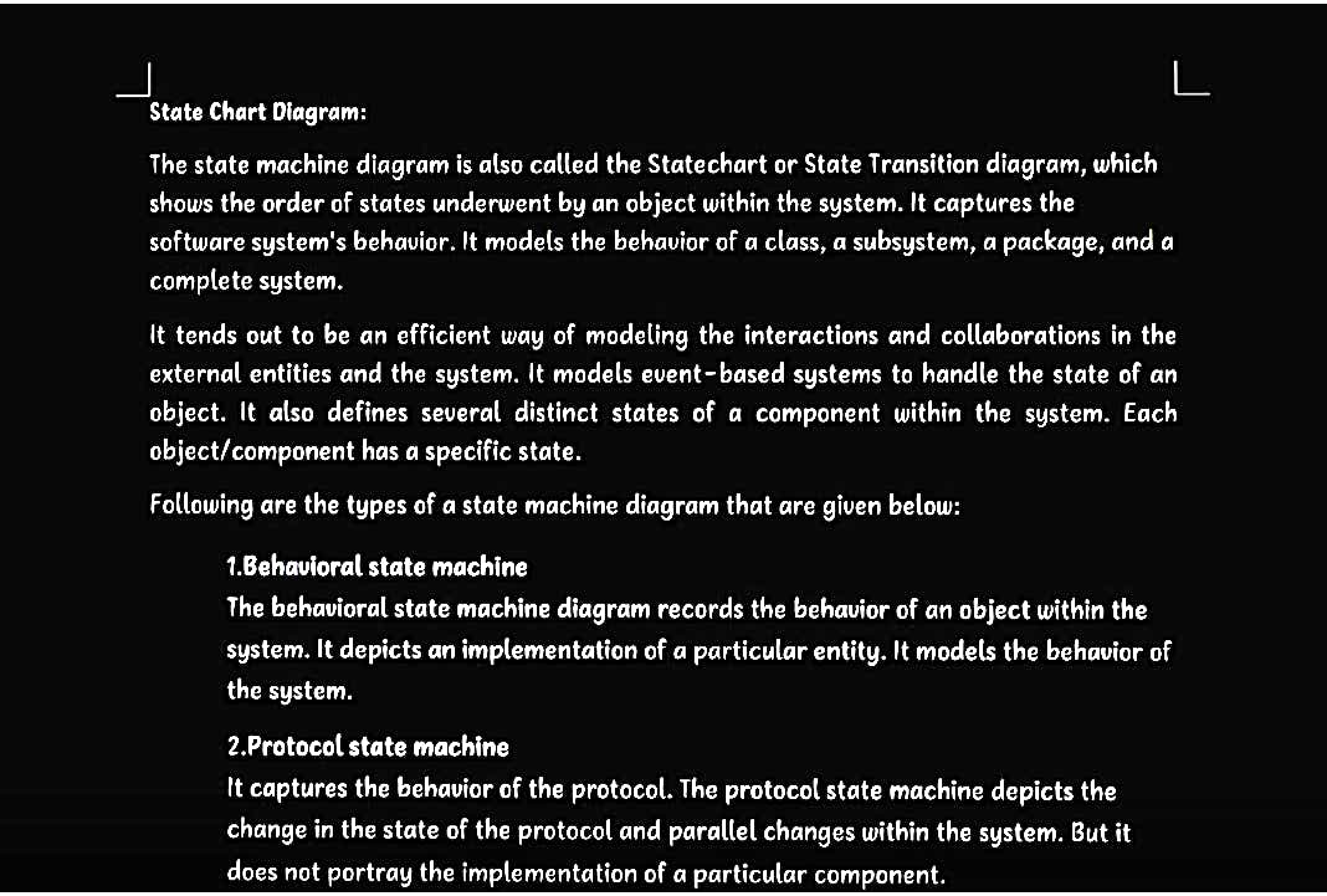
1. **Objects:** The representation of an object is done by an object symbol with its name and class underlined, separated by a colon.
   In the collaboration diagram, objects are utilized in the following ways:
   - The object is represented by specifying their name and class.
   - It is not mandatory for every class to appear.
   - A class may constitute more than one object.
   - In the collaboration diagram, firstly, the object is created, and then its class is specified.
   - To differentiate one object from another object, it is necessary to name them.
2. **Actors:** In the collaboration diagram, the actor plays the main role as it invokes the interaction. Each actor has its respective role and name. In this, one actor initiates the use case.
3. **Links:** The link is an instance of association, which associates the objects and actors. It portrays a relationship between the objects through which the messages are sent. It is represented by a solid line. The link helps an object to connect with or navigate to another object, such that the message flows are attached to links.
4. **Messages:** It is a communication between objects which carries information and includes a sequence number, so that the activity may take place. It is represented by a labeled arrow, which is placed near a link. The messages are sent from the sender to the receiver, and the direction must be navigable in that particular direction. The receiver must understand the message.





Ex:

Window: UserInterface
Object
1: makeReservation
Message
aChain: TrainChain
2: makeReservation
3: isSeat
aTrain: Train
aReservation: Reservation
aNotice : Confirmation
SelfLink
4: isSeat



State Chart Diagram:

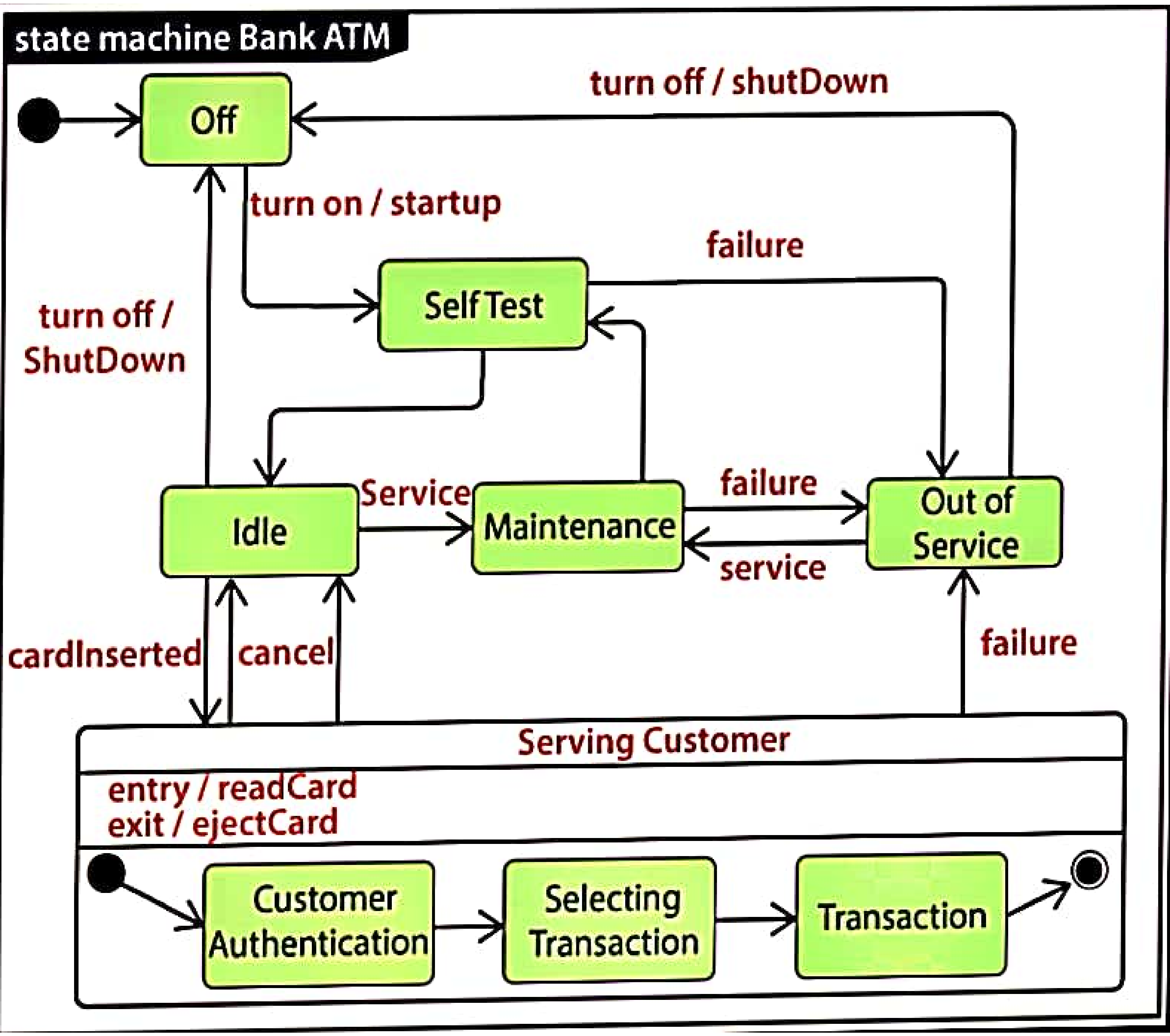
The state machine diagram is also called the Statechart or State Transition diagram, which shows the order of states underwent by an object within the system. It captures the software system's behavior. It models the behavior of a class, a subsystem, a package, and a complete system.

It tends out to be an efficient way of modeling the interactions and collaborations in the external entities and the system. It models event-based systems to handle the state of an object. It also defines several distinct states of a component within the system. Each object/component has a specific state.

Following are the types of a state machine diagram that are given below:

1.Behavioral state machine
The behavioral state machine diagram records the behavior of an object within the system. It depicts an implementation of a particular entity. It models the behavior of the system.

2.Protocol state machine
It captures the behavior of the protocol. The protocol state machine depicts the change in the state of the protocol and parallel changes within the system. But it does not portray the implementation of a particular component.


state machine Bank ATM
Off
turn off / shutDown
turn on / startup
failure
Self Test
turn off / ShutDown
Idle
Service
Maintenance
failure
Out of Service
service
cardInserted
cancel
failure
Serving Customer
entry / readCard
exit / ejectCard
Customer Authentication
Selecting Transaction
Transaction

UML Activity Diagram

In UML, the activity diagram is used to demonstrate the flow of control within the system rather than the implementation. It models the concurrent and sequential activities.

The activity diagram helps in envisioning the workflow from one activity to another. It put emphasis on the condition of flow and the order in which it occurs. The flow can be sequential, branched, or concurrent, and to deal with such kinds of flows, the activity diagram has come up with a fork, join, etc.

It is also termed as an object-oriented flowchart. It encompasses activities composed of a set of actions or operations that are applied to model the behavioral diagram.

Notation of an Activity diagram

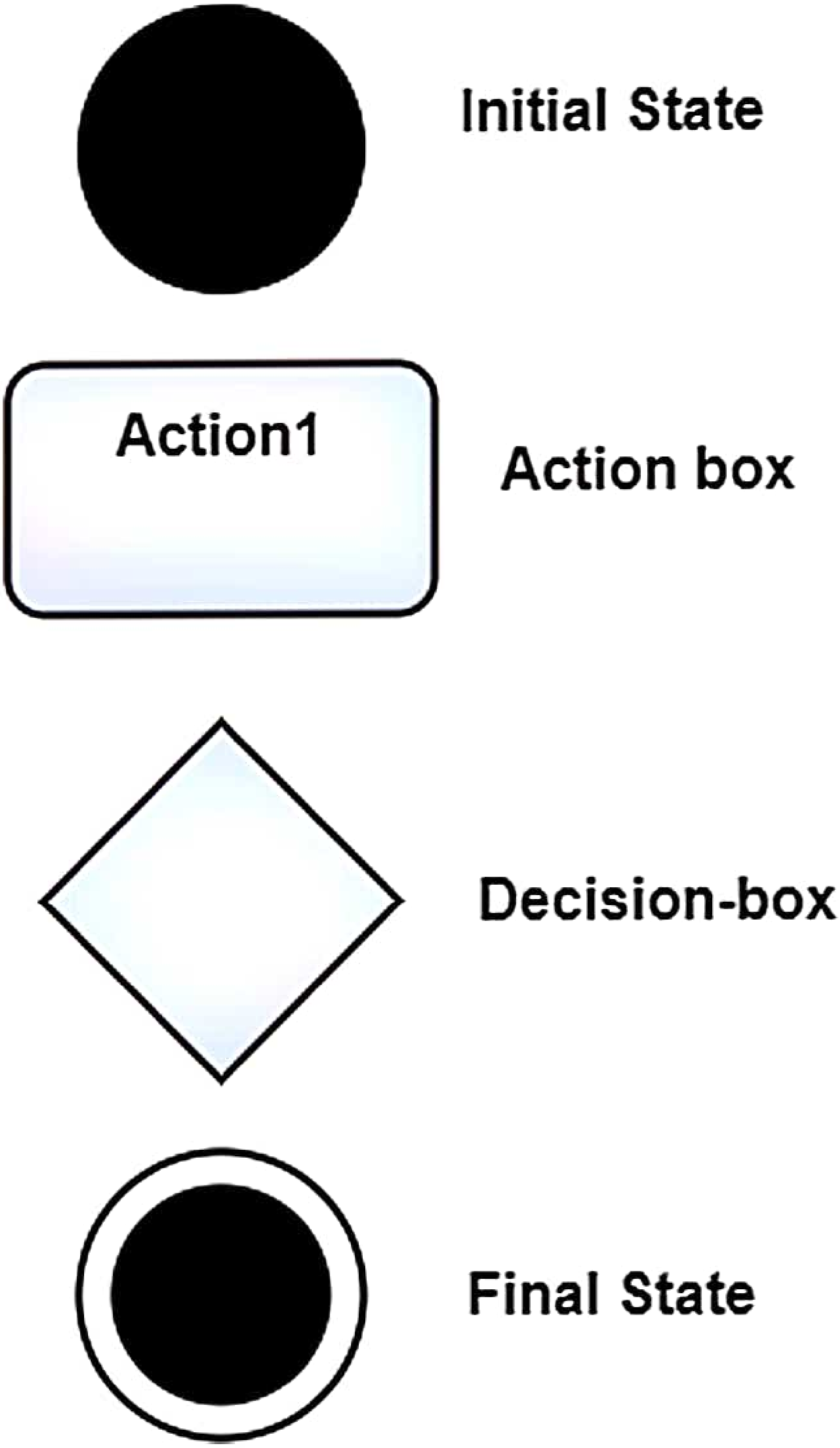
Activity diagram constitutes following notations:

Initial State: It depicts the initial stage or beginning of the set of actions.

Final State: It is the stage where all the control flows and object flows end.

Decision Box: It makes sure that the control flow or object flow will follow only one path.

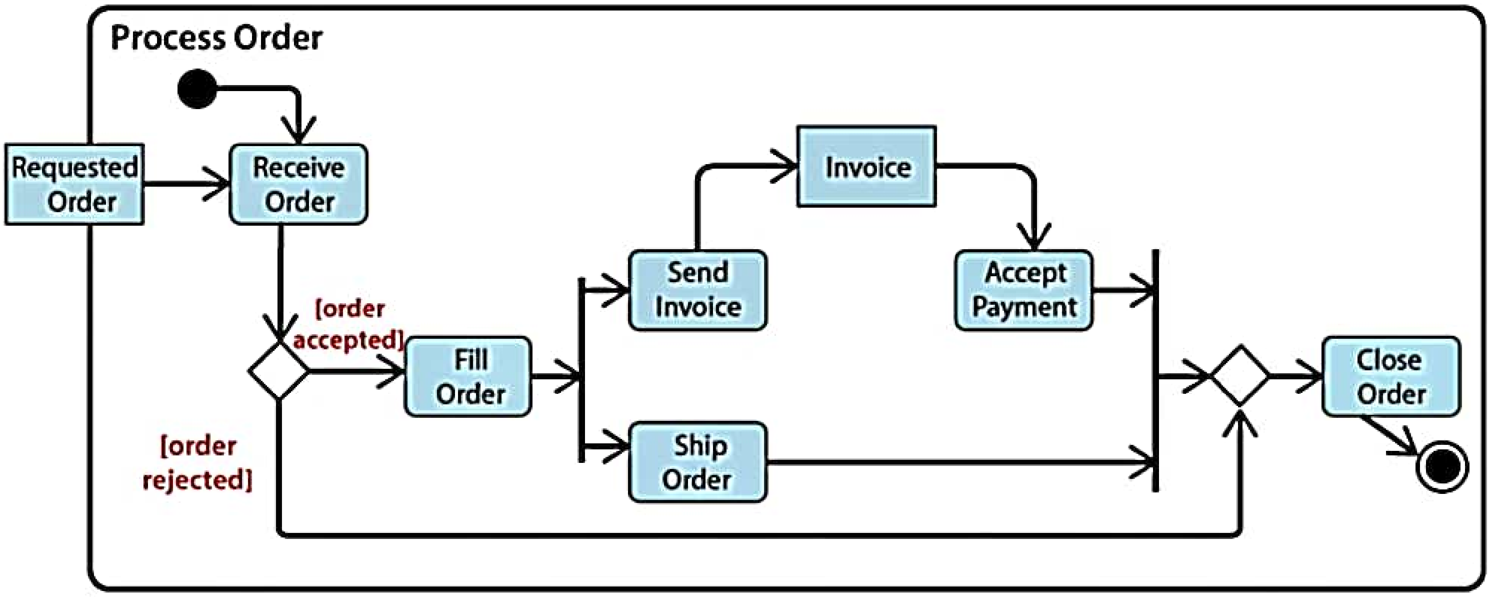
Action Box: It represents the set of actions that are to be performed.


Initial State
Action1
Action box
Decision-box
Final State

Example of an Activity Diagram

An example of an activity diagram showing the business flow activity of order processing is given below.

Here the input parameter is the Requested order, and once the order is accepted, all of the required information is then filled, payment is also accepted, and then the order is shipped. It permits order shipment before an invoice is sent or payment is completed.



Introduction to UML:

The *Unified Modeling Language* (UML) is "a standard language for writing software blueprints. UML may be used to visualize, specify, construct, and document the artifacts of a software-intensive system".

In other words, just as building architects create blueprints to be used by a construction company, software architects create UML diagrams to help software developers build the software. If you understand the vocabulary of UML (the diagrams' pictorial elements and their meanings), you can much more easily understand and specify a system and explain the design of that system to others.

Grady Booch, Jim Rumbaugh, and Ivar Jacobson developed UML in the mid- 1990s with much feedback from the software development community.

UML 2.0 provides 13 different diagrams for use in software modeling.

Goals of UML

- Since it is a general-purpose modeling language, it can be utilized by all the modelers.
- UML came into existence after the introduction of object-oriented concepts to systemize and consolidate the object-oriented development, due to the absence of standard methods at that time.
- The UML diagrams are made for business users, developers, ordinary people, or anyone who is looking forward to understand the system, such that the system can be software or non-software.
- Thus it can be concluded that the UML is a simple modeling approach that is used to model all the practical systems.

Characteristics of UML

The UML has the following features:

- It is a generalized modeling language.
- It is distinct from other programming languages like C++, Python, etc.
- It is interrelated to object-oriented analysis and design.
- It is used to visualize the workflow of the system.

- It is a pictorial language, used to generate powerful modeling artifacts.