Assignment 2
Object Oriented Analysis and Design.
Group 5

Ratnesh Pawar	**Rohan Sonawane**	**Mayank Sharma**
Abhishek Dhale	**Neil Duraiswami**	**FNU Sumit Kumar**

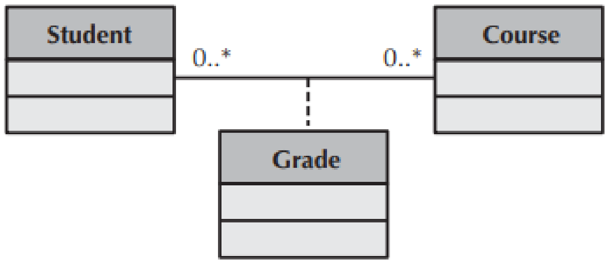
1.
   a. An association class is a class that is a part of an association connection between two other classes. There are times when a relationship itself has associated properties, especially when its classes share a many-to-many relationship. In these cases, a class called an association class is formed, which has its own attributes and operations.

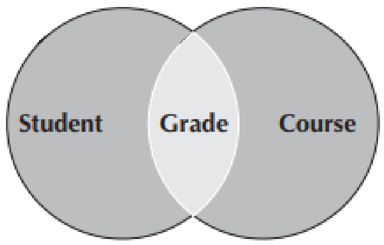
      Ex. The "married to" relationship links a Man class and a Woman class, and the marriage date must be stored. But rather than being explicitly owned by the Man object or the Woman object, that data is connected to the relationship. To record such data in such cases, an extra association class—such as a Marriage class—may be established.

   b. The purpose of a class diagram is to show relationships or associations that classes have with one another. They are presented on the diagram by drawing lines between classes.

      Example of Association Class

      Since a grade may only exist at the intersection of the Student and Course classes, the Grade notion is actually an intersection of these two concepts.





2.

Object:

- An object is an instance of a class.
- It represents a specific, individual entity or instance in a program.
- Objects have attributes (data) and methods (functions) associated with them.

Class:

- A class is a blueprint or template for creating objects.
- It defines the structure and behavior that its objects will have.
- Classes can be thought of as user-defined data types.

Method:

- A method is a function, or a behavior associated with a class.
- It defines what actions or operations can be performed on the objects of that class.
- Methods can access and manipulate the attributes of an object.

Attribute:

- An attribute is a data field or property that belongs to an object.
- It represents the characteristics or state of an object.
- Attributes are defined in the class and hold values specific to each object.

Superclass:

- A superclass is a class from which other classes inherit properties and behaviors.
- It is also called a base class or parent class.
- Subclasses can inherit attributes and methods from the superclass.

Subclass:

- A subclass is a class that inherits attributes and methods from a superclass.
- It can also add its own unique attributes and methods.
- Subclasses are also referred to as derived classes.

Concrete Class:

- A concrete class is a class that can be instantiated to create objects.
- It has complete implementations of its methods and can be used directly.

Abstract Class:

- An abstract class is a class that cannot be instantiated on its own.
- It may have some methods without implementations (abstract methods).
- It's meant to be subclassed, and its subclasses are expected to provide implementations for the abstract methods.

3.

a.

In object-oriented design, three important types of relationships exist **aggregation, generalization, and association**. Each type serves a unique purpose in modeling the structure and behavior of a system.

Here are two examples of each type of relationship and how they can be depicted on a class diagram:

Aggregation:
Aggregation is a "whole part" relationship where one class (the whole) contains or is composed of other classes (the parts). However, the lifetimes of the whole and parts are independent.

Example 1: A University and Department.
A university has several departments, but a department can exist independently of the university.
Example 2: A Computer and Keyboard.
A computer has a keyboard, but the keyboard can exist separately from the computer.

Depiction on a class diagram: Aggregation is depicted by a hollow diamond shape at the whole class end and a line connecting it to the part class.

Generalization (Inheritance):
Generalization represents an "is-a" relationship and is used to depict inheritance in object-oriented design.

Example 1: A Vehicle and Car.
A car is a type of vehicle.
Example 2: An Animal and Dog.
A dog is a type of animal.

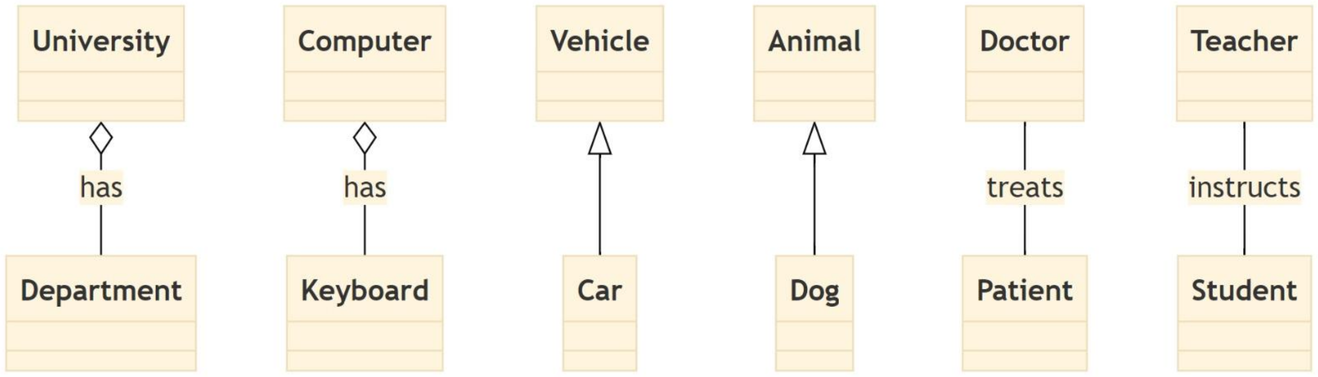
Depiction on a class diagram: Generalization is depicted by a hollow triangle shape on the superclass end and a line (or a tree of lines) connecting it to the subclass.

Association:
The association represents a bi-directional, structural link between two classes.

Example 1: A Doctor and Patient.
A doctor treats a patient, and a patient is treated by a doctor.
Example 2: A Teacher and Student.
A teacher instructs students, and a student is taught by a teacher.

Depiction on a class diagram: Association is depicted by a line connecting the two associated classes. The multiplicity (like 1..*, 1..1) can be added at both ends to indicate the number of objects involved in the relationship.

Figure: Aggregation, Generalization, and Association depiction on the class diagram.



b.

Figure: A patient must be assigned to only one doctor, and a doctor can have one or more patients.

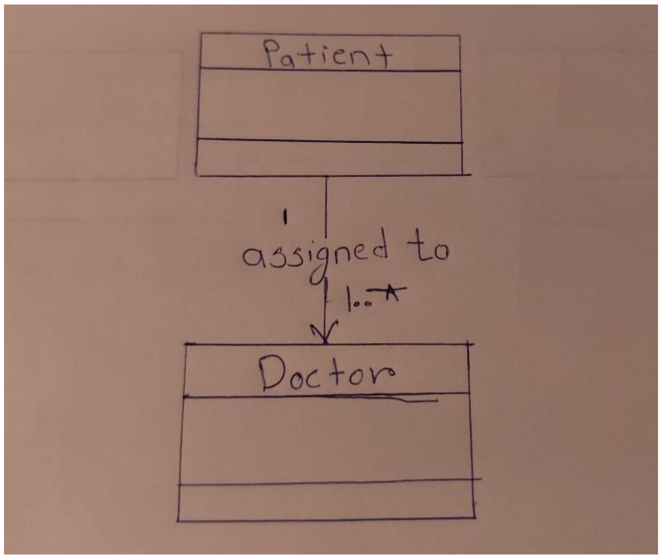


Figure: An employee has one phone extension, and a unique phone extension is assigned to an employee.

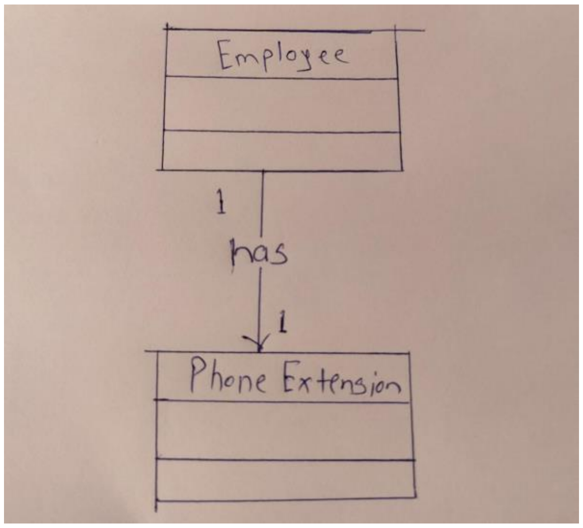


Figure: A movie theater shows at least one movie, and a movie can be shown at up to four other movie theaters around town.

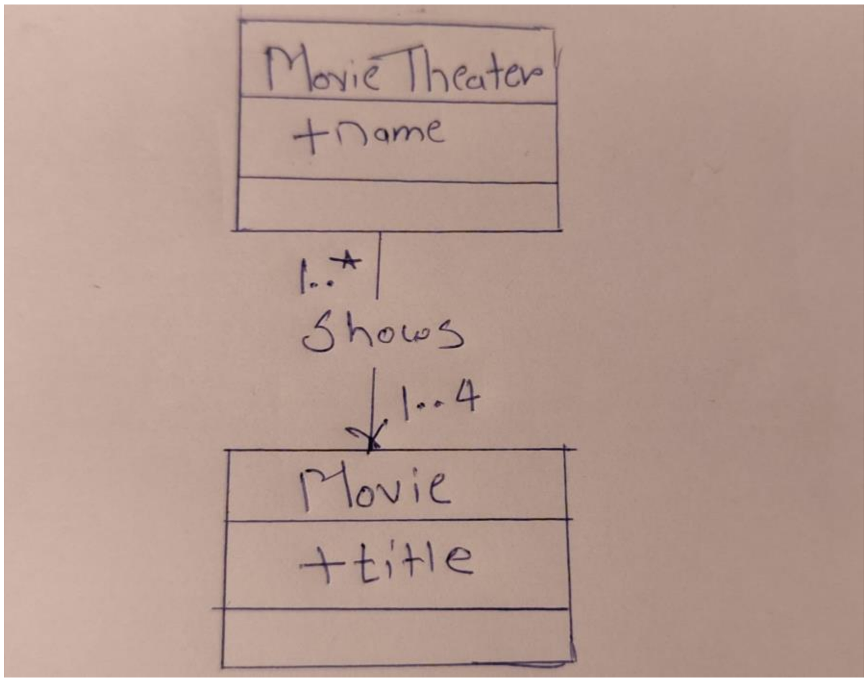
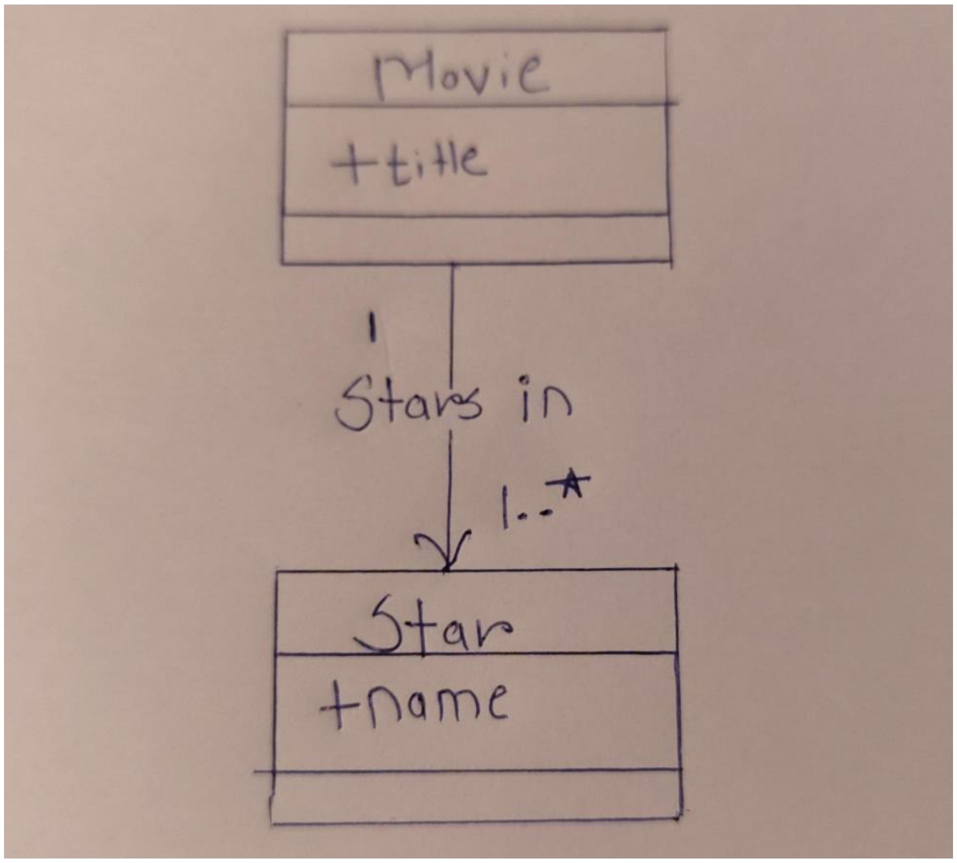


Figure: A movie either has one star, two co-stars, or more than 10 people starring together. A star must be in at least one movie.



4.

The main role of the structural models is to produce a vocabulary that can be used by analysts and the user to make communications effectively.

The different structural models are related through different relationships. For example, an association is a relationship between two classifiers, such as classes or use cases, that describes the reasons for the relationship and the rules that derive the relationship. Three structural modeling representations are CRC Cards, Class diagrams, and Object diagrams.

Let's discuss a little bit about how these three related to each other's.

- Class diagrams can be used to generate an Object diagram, which shows the instances of the classes and their values at a specific point in time.
- Similarly, an Object diagram can also be derived from a Class diagram by applying some constraints or making some changes to the classes.
- CRC card can show the responsibilities and collaborations of each class of class diagram to fulfill a use case.

The relationship between different structural models does have a significant impact on the verification and validation of the model. In terms of Structural models, Verification is the process of checking whether the model or diagram is built correctly, according to the defined constraints and rules. Validation is the process of checking whether the model or diagram is an accurate representation of the real system or problem domain.

In the case of Structural modeling, the verifications and validations are done during formal review meetings using a walkthrough approach in which the analyst presents the model to a team of developers and users. Developer's needs to justify the reasons behind the inclusion of attributes, operations and relationships associated with the classes. Otherwise, it will be meaningless. There are several things that must be considered like, each class should be linked back to at least one-use case.

These are some rules to verify and validate structural models:

- Each CRC card is associated with a class in class diagram.
- Responsibilities on the front of the card are included as operations on the class diagram.
- Collaborators on the front of the card imply a relationship on the back of the card.
- Attributes on the back of the card are listed as attributes on the class diagram.
- Attributes on the back of the CRC card each have a data type.
- Relationships on the back of the card must be properly depicted on the class diagram.
- Association classes are used only to include attributes that describe a relationship.
- Specific representation rules must be enforced, such as a class cannot be a subclass itself.

If the structural model does not follow these rules, then it may contain ambiguity, errors, inconsistencies, or incompleteness that can affect the quality and correctness of the system.

5. The relationship between structural models and functional models is essential in the process of verification and validation of these models. Here's how they are related.

   Structural Models:

   - **Representation:** Structural models depict the physical or logical components of a system and how they are organized or interconnected.
   - **Focus:** They focus on the "what" of a system, detailing its parts and their relationships.
   - **Examples:** Block diagrams, data flow diagrams, and UML class diagrams.

   Functional Models:

   - **Representation:** Functional models describe how a system or object behaves or functions, emphasizing processes, interactions, and behavior.
   - **Focus:** They deal with the "how" of a system, specifying its dynamic aspects, inputs, processes, and outputs.
   - **Examples:** Flowcharts, state transition diagrams, and use case diagrams.

   Relationship between Structural and Functional Models:

   The relationship lies in the mapping between structural components and their corresponding functional behavior. In other words, it's about associating the **"what" (structural elements)** with the **"how" (functional aspects)** they perform.

   - In a UML class diagram (structural model), classes and their relationships are represented, detailing the structural components.
   - In a use case diagram (functional model), use cases depict the functional behavior. Each use case is linked to one or more classes from the class diagram. This connection maps structural elements (classes) to their functional roles (use cases).

   Verification and Validation:

   - **Verification:** This process ensures that a model accurately represents the system or object it intends to depict. For structural models, verification checks that the components and connections are correctly represented. For functional models, it involves confirming that the functional behavior aligns with the desired system operation.
   - **Validation:** This step confirms whether the model serves its intended purpose. For structural models, validation ensures that the chosen components and relationships are suitable for solving the problem at hand. For functional models, it checks if the functional behavior aligns with the actual requirements and goals of the system.

6.

a.

- Behavioral modeling focuses on representing how a system behaves dynamically, typically through diagrams like activity diagrams, state transition diagrams, and sequence diagrams.
- Structural modeling, on the other hand, deals with the static elements of a system, such as its components, attributes, relationships, and organization, often depicted in diagrams like class diagrams and entity-relationship diagrams.
- The relationship lies in the fact that structural models provide **the foundational components that are often used within behavioral models**. For example, in an activity diagram (behavioral), you might depict actions involving classes or objects from a class diagram (structural).

b.

- A use case is a description of a specific functionality or interaction within a system from the perspective of an external user (an actor).
- An activity diagram is used to represent the dynamic flow of activities and actions within a system.
- The relationship between a use case and an activity diagram is that a use case can be depicted as a series of activities in an activity diagram. Each use case typically corresponds to an activity or a set of activities that show how the user interacts with the system to achieve a particular goal.

c.

- A use case defines a specific interaction or functionality from the perspective of an actor.
- A generic sequence diagram is a more abstract representation of interactions between objects but is not tied to a specific use case.
- The relationship lies in the fact that a use case can be represented more concretely in a sequence diagram. A sequence diagram provides a detailed view of how objects interact during the execution of a specific use case.

d.

- A use case scenario is a specific instance or flow of events related to a particular use case, detailing how a user interacts with the system to accomplish a specific task.
- An activity diagram can represent the dynamic flow of actions within a use case, including different scenarios.
- Use case scenarios can serve as the basis for creating the content of an activity diagram. The activities within the activity diagram can be derived from the steps or actions outlined in the use case scenario.

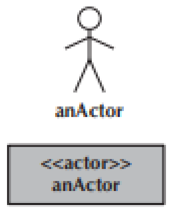
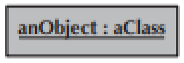
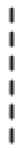
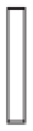
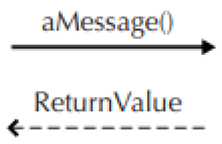
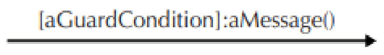
e.

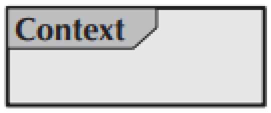
- An instance sequence diagram shows the interactions between objects (instances) as they execute a specific scenario or use case.
- A use case scenario is a textual description of how a specific use case is executed.
- An instance sequence diagram is a graphical representation of the same scenario, illustrating how different object instances interact during the execution of that scenario.

7.

a.

Sequence diagrams are a type of UML (Unified Modelling Language) diagram used to visualize the interactions and communication among objects or components within a system. The main building blocks or elements of a sequence diagram include:

Term and Definition	Symbol
Actor: • An actor is an external entity, which can be either a person or a system, that interacts with the system under consideration. • Actors participate in a sequence diagram by sending and/or receiving messages to and from objects within the system. • In the diagram, actors are typically positioned at the top and represented as either stick figures (by default) or as rectangles containing the <<actor>> stereotype if nonhuman actors are involved.	
Object: • Objects represent instances of classes within the system. • They participate in a sequence diagram by sending and/or receiving messages. • Objects are positioned at the top of the diagram.	
Lifeline: • A lifeline represents the lifespan of an object throughout the sequence of interactions. • It is depicted as a vertical dashed line extending downwards from an object, with an 'X' marking the point where the object ceases to interact in the sequence.	
Execution Occurrence: • Execution occurrences are elongated rectangles placed atop a lifeline. • They indicate the specific points in time when an object is actively engaged in sending or receiving messages during the sequence.	
Message: • Messages are a means of conveying information between objects within the system. • An operation call message is labelled with the message itself and is represented by a solid arrow pointing from the sender to the receiver. • A return message, denoting the value being returned, is indicated by a dashed arrow.	
Guard Condition: • Guard conditions are conditions or tests that must be satisfied for a message to be sent within the sequence.	

• They represent constraints on the communication between objects and are typically used to model decision points or branches in the sequence of interactions.	
Object Destruction: • Object destruction is represented by placing an 'X' symbol at the end of an object's lifeline. • This symbol indicates the termination or the point at which the object goes out of existence within the sequence.	**X**
Frame: • A frame is used to provide context and structure within a sequence diagram. • It serves to enclose and group a set of related interactions or messages, helping to organize and clarify the diagram. • Frames are often used to represent various aspects of the system or to denote specific scenarios or contexts within the sequence of interactions.	

b.

In a sequence diagram, showing the termination or destruction of a temporary object is critical for accurately modeling a system's dynamic behavior.

Place an **'X'** symbol at the end of the object's lifeline to explicitly show when the temporary object is to be destroyed or cease to exist. This 'X' symbol clearly marks the object's termination point within the sequence.

Consider the following scenario involving a shopping cart object in a Web commerce application. The shopping cart is temporarily used to collect line items for an order. When an order is confirmed, the shopping cart becomes irrelevant and is no longer required. To demonstrate the shopping cart's end, place the 'X' symbol exactly where it is destroyed on the lifeline.

c.

Lifelines in a sequence diagram do not always extend the entire length of the page. The exact length of a lifeline is determined by the context as well as the specific objects or actors represented in the diagram. Lifelines only last if they are involved in the depicted sequence of interactions.

Here's how it works:

- The existence and activity of objects or actors within the scope of the sequence being illustrated are represented by lifelines. They begin when an object or actor is introduced into the sequence and end when their participation in that sequence is complete. When an object's or actor's role in the interaction is finished, their lifeline is terminated.
- In practice, the length of lifelines can differ between objects or actors within the same diagram. Some objects have short lifelines and participate in only a few interactions, whereas others have

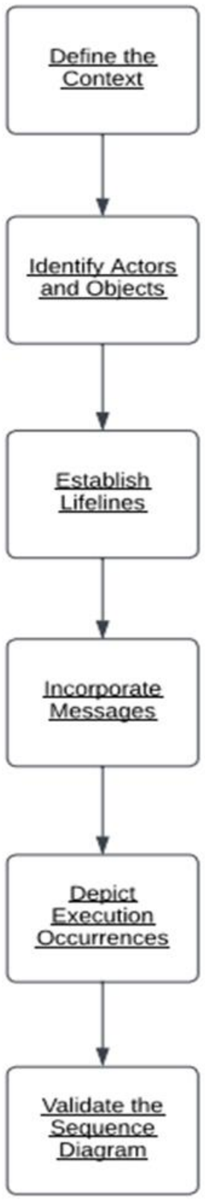
longer lifelines that extend across multiple interactions or the entire diagram if they remain active throughout the sequence.

- Lifelines can be explicitly terminated to indicate when an object is destroyed or ceases to exist, as indicated by an 'X' symbol at the end of the lifeline. If an object is meant to last longer than the sequence depicted, its lifeline will continue downward from the last interaction without an 'X' symbol.

8.

a.

Step 1: Define the Context


Define the Context
Identify Actors and Objects
Establish Lifelines
Incorporate Messages
Depict Execution Occurrences
Validate the Sequence Diagram

- Begin the sequence diagram creation process by defining the context in which it will be used. This context can take the form of a system representation, a specific use case, or a selected scenario nested within a use case.
- Enclose the diagram with a labelled frame to visually represent the context. This frame serves as a defining boundary, defining the scope of the interactions being modeled.
- A single use-case scenario is typically used as the immediate context. In complex systems, however, it may be advantageous to generate multiple instance-specific sequence diagrams to investigate various situational contexts.

Step 2: Identify Actors and Objects

- Continue by identifying the system's key participants, which include actors (representing external entities) and objects (corresponding to internal components or classes that actively participate in the interactions).
- Typically, actors are identified during the development of the functional model, whereas objects are identified during the development of the structural model.
- The sequence diagram is built on the foundational elements of actors and objects.

Step 3: Establish Lifelines

- Devote the third step to configuring lifelines for each object identified. A lifeline is a vertical, stippled line that runs beneath the relevant class or object.

- Denote the end of an object's existence within the sequence by appending an 'X' symbol at the exact point of deactivation or discontinuation.

Step 4: Incorporate Messages

- Continue by inserting messages into the diagram to represent interactions and communication between objects.
- Messages appear as arrows that represent the flow of information between objects. The direction of these arrows indicates the communication direction.
- To depict the chronological sequence of events, arrange the messages hierarchically from top to bottom.
- Enclose message parameters in parentheses next to the message nomenclature. Return messages, which represent messages' responses, do not need to be explicitly depicted in the diagram.

Step 5: Depict Execution Occurrences

- Increase the expressiveness of the diagram by adding execution occurrences to the lifelines of objects.
- Execution occurrences appear as thin rectangular boxes superimposed on the lifelines, indicating when classes are actively transmitting or receiving messages.
- These occurrences provide a visual representation of temporal aspects, shedding light on the timing and duration of object interactions.

Step 6: Validate the Sequence Diagram

- Finish the sequence diagram creation process with a thorough validation stage to ensure that the diagram accurately represents the underlying process.
- Ensure that the diagram accurately depicts all required steps and interactions, ensuring a complete depiction of the system's behavioral dynamics.

b.

When creating a sequence diagram, it is critical to follow established guidelines to improve the diagram's clarity and ease of interpretation. following are the guidelines, which are considered best practices in the creation of sequence diagrams:

- **Sequential Message Ordering:**
  Whenever possible, arrange messages not only top-to-bottom but also, and most importantly, left-to-right. This arrangement corresponds to the typical reading patterns observed in Western cultures, in which information is typically processed from left to right and top to bottom. To achieve this alignment, actors and objects should be arranged along the top of the diagram in the order in which they appeared in the use-case scenario.

- **Uniform Naming Conventions:**
  When an actor and an object conceptually represent identical entities, even if one is located within the software system and the other in an external context, it is best to use the same nomenclature for both. This assumes that these entities are represented in both the use-case diagram, as actors, and the class diagram, as classes. The overarching principle here is that consistent naming eliminates ambiguity and confusion.

- **Initiator Positioning:**
  The scenario's initiator, whether an actor or an object, should be positioned as the diagram's leftmost entity. This recommendation essentially extends and refines the first guideline, emphasizing the pivotal role of the initiator. This arrangement improves the diagram's readability and visual flow.

- **Object Naming in Multiplicity:**
  In cases where multiple objects of the same class exist within the sequence, it is best to give each object a unique name in addition to indicating its class. This practice helps to distinguish and disambiguate objects that share a common class designation. If multiple objects of the "Parent" class are present, for example, individual names should be assigned to each object to improve clarity. However, for objects with distinct attributes or properties, the use of the class name alone may suffice. A "Child" object, for example, may not require a separate name, and a colon preceding the class name can effectively denote its presence.

- **Prudent Use of Return Values:**
  The use of return values in a sequence diagram should be done with caution. Return values should be explicitly displayed only when their significance is unclear. Excessive representation of return values can increase diagram complexity and potentially obscure the diagram's core message. It is best to display return values selectively, keeping only those that provide relevant information for the diagram's interpretation.

- **Message and Return Value Alignment:**
  It is recommended that message names and return values be aligned close to the respective arrowheads of the message and return arrows to improve the diagram's readability and interpretability. This alignment facilitates a more coherent understanding of the diagram's dynamics by streamlining the cognitive process for interpreting both messages and their associated return values.

9.

a.

State transitions in a behavioral state machine are triggered by specific events that cause an object's state to change.

These occurrences can be classified as follows:

Signal Events:

- Signal events are asynchronous occurrences in which one object dispatches and another object receives a named entity.
- These events are initiated by objects within the system and can result in state transitions when the receiving object acknowledges the signal.
- Signal events can have properties, operations, relationships, and instances. Attributes can be used as parameters and are frequently associated with actions such as state transitions or message transmission.

Call Events:

- Call events represent the synchronous start of an operation. An object typically invokes a specific operation on another object.
- Such occurrences are synchronous, which means that the caller is waiting for a response from the callee while the operation is being performed.
- Call events are like method calls in programming and are essential to synchronous interactions.

Time Events:

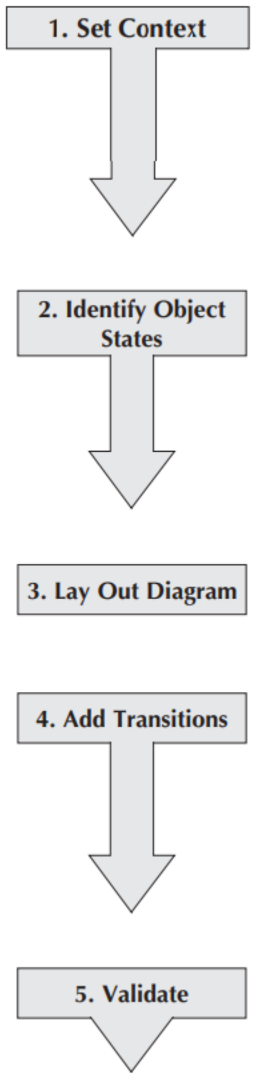
- Time events denote the passage of time and act as triggers for state transitions at predefined time intervals or specific points in time.
- They are defined by expressions such as 'after' followed by a time duration that can be simple or complex.
- Time events are frequently used to model time-dependent behaviors like timeouts and delays.

Change Events:

- Changing events denote changes in the state of an object or the fulfillment of specific conditions.
- These events are triggered when a specified Boolean condition becomes true, indicating that a state transition should occur.
- Change events are useful in modeling state changes caused by changes in object attributes or specified conditions.

These event types form the basis for initiating state transitions within a behavioral state machine. One or more of these event classifications may be used to efficiently record and represent state transitions in a coherent and structured way, based on the system's characteristics and requirements.

b.



Step 1: Set Context

- The first step in developing a behavioral state machine is determining the context in which it will be used.
- Behavioral state machines are typically generated from a class diagram for individual classes.
- The context is denoted by a label inside of the diagram's frame that specifies the class or classes being modeled. It can also refer to a specific subsystem or the entire system.

Step 2: Identify Object States

- The various states that an object will go through during its existence are identified in this step.
- This involves defining an object's initial and final states as well as recognizing the states that an object can take.
- This step's information is derived from use-case descriptions, user interactions, and requirements-gathering techniques.
- Figuring out states is frequently aided by writing out the chronological sequence of events that occur during the lifecycle of an object.

Step 3: Layout Diagram

- The third step entails determining the order in which an object transitions between states over time.
- The identified states are positioned onto the behavioral state machine diagram within the order of state transitions, from left to right.

Step 4: Add Transitions

- Transitions between the states are identified and labeled in this step, along with the events, actions, and guard conditions that trigger or influence the transitions.
- Events are the triggers that cause an object to change its state. They are frequently associated with actions, which are usually operations performed by the object.
- Guard conditions are sets of conditions that must be met before a transition can take place.
- Transitions are drawn to connect the relevant states and are labeled with the corresponding event, action, or guard condition.

Step 5: Validate the State Machine

- The final step is to validate the behavioral state machine to ensure that each state is accessible and that all states can be left except the final ones.
- If a state is discovered to be unreachable, it indicates that either a transition is missing, or the state was incorrectly identified.
- Final states, which represent the end of an object's lifecycle, can be regarded as "dead ends" and are not expected to be followed by transitions.

c.

The following are some guidelines to follow when creating a behavioral state machine.

Model Objects with Changing Behavior:

- Design a behavioral state machine for objects whose behavior varies greatly depending on their state.
- Avoid creating state machines for objects whose behavior remains constant regardless of their state because they are too simple for this modeling approach.

Initial and Final State Placement:

- To align with Western cultures' left-to-right and top-to-bottom reading conventions, place the initial state in the top left corner of the diagram.
- Place the final state in the diagram's bottom right corner. This design promotes a natural flow of reading.

Use Descriptive State Names:

- State names should be simple, easy to remember, and evocative of the state's purpose or meaning.
- In the example provided, for a patient object, the state names include Entering, Admitted, Under Observation, and Released.

Address Black Hole and Miracle States:

- Recognize and challenge the existence of black holes and miracle states within the state machine.

- Black hole states, in which an object enters but never exists, should be evaluated, and may have to be treated as final states.
- Miracle states, in which an object exists without ever entering, should be considered as initial states.

Ensure Mutually Exclusive Guard Conditions:
- Guard conditions, which establish state transition conditions, must be mutually exclusive.
- Avoid overlapping guard conditions because they can cause ambiguity in determining which state to transition to.
- For example, guard conditions [Diagnosis = Healthy] and [Diagnosis = Unhealthy] should not overlap.
- In cases such as [x >= 0] and [x = 0], overlapping occurs when x = 0, causing confusion.

Associate Transitions with Messages and Operations:
- Every state transition must be accompanied by a message and an operation.
- This association ensures that the object's state changes in accordance with the events or actions.
- To avoid unintended consequences, analysts should double-check that each transition is indeed connected with a message and operation.

10.

a. **CRUDE** analysis aids in understanding interactions between objects in an object-oriented system. The acronym "CRUDE" stands for:

- **C**: Create | - **R**: Read or Reference | - **U**: Update | - **D**: Delete | - **E**: Execute

A CRUDE matrix is just a table with actors or classes represented in both the rows and the columns. The interactions between actors or classes from different rows and columns are shown in each matrix cell. If a "Receptionist" actor, for example, produces an instance of the "Appointment" class, the letter "C" will appear in the matrix where the "Receptionist" row and the "Appointment" column intersect.

Multiple functions are accomplished using the CRUDE matrix. In addition to providing an overview of system-wide connections, it also aids in validating interactions, locating groups of closely linked classes, and spotting classes with potentially complicated life cycles.

Example: **CRUDE Matrix for E-commerce System:**

	Product	**Order**	**Cart**	**Payment System**	**Review**
Customer	R,U	C,R	C,R,U,D	E	C,R,U
Product	R,U	R	R	-	R
Order	R	R,U,D	R	E	-
Cart	R	R	R,U	E	-
Payment System	-	R	R	R,U	-
Review	R	-	-	-	R,U

Customer:
Reads (R) and updates (U) product details.
Creates (C) and reads (R) orders.
Manages cart by creating (C), reading (R), updating (U), and deleting (D) items in it.
Executes (E) payments.
Writes (C), reads (R), and updates (U) product reviews.

Product:
Product details can be read (R) and occasionally updated (U).
Referenced in orders (R) and charts (R).
Reviews can be read (R) for a product.

Order:
Reads (R) product details.
Order details can be read (R), updated (U), and deleted (D).
Executes (E) payments through the payment system.

Cart:
Reads (R) product details.
References orders (R).
Items in cart can be read (R), updated (U), and checked out using payment system (E).

Payment System:
Processes payments for orders (R) and carts (R).
Transaction details are read (R) and updated (U).

Review: Product reviews can be read (R) and updated (U).

b.

Actor: Use Case Model
Association: Class and Object Diagram
Class: Class Diagram
Extends: Use Case Model
Final State: State chart Diagram
Guard Condition: State chart Diagram, Sequence Diagram, Activity Diagram
Initial State: State chart Diagram
Links: Object Diagram
Message: Sequence Diagram, Communication Diagram
Multiplicity: Class Diagram
Object: Object Diagram, Sequence Diagram and Communication Diagram
State: State chart Diagram
Transition: State Chart Diagram and Activity Diagram
Update Operation: Class Diagram and Sequence Diagram