OBJECT ACTIVITY ANALYSIS SESSION 2

SESSION OBJECTIVE

- Partitions
- Pins in activity diagram
- When to Use Activity Diagram
- UML Diagram
- Interaction Diagram
- Sequence Diagram
- Sequence Diagram Frames
- Introduction to Object State
- State MACHINE DIAGRAM

PARTITIONS

LEARN MORE

PARTITIONS

What is Partition ?

In the context of an activity diagram, partitions, also known as swimlanes, are graphical elements used to organize and categorize activities and actors involved in a system or process. Partitions visually divide the activity diagram into horizontal or vertical sections, each representing a specific participant, role, or entity involved in the system.
Partitions help in depicting the flow of activities and the interaction between different entities or actors in a clear and organized manner. They provide a visual representation of how activities are distributed among different participants and can be used to show responsibilities, ownership, or the system's overall structure.

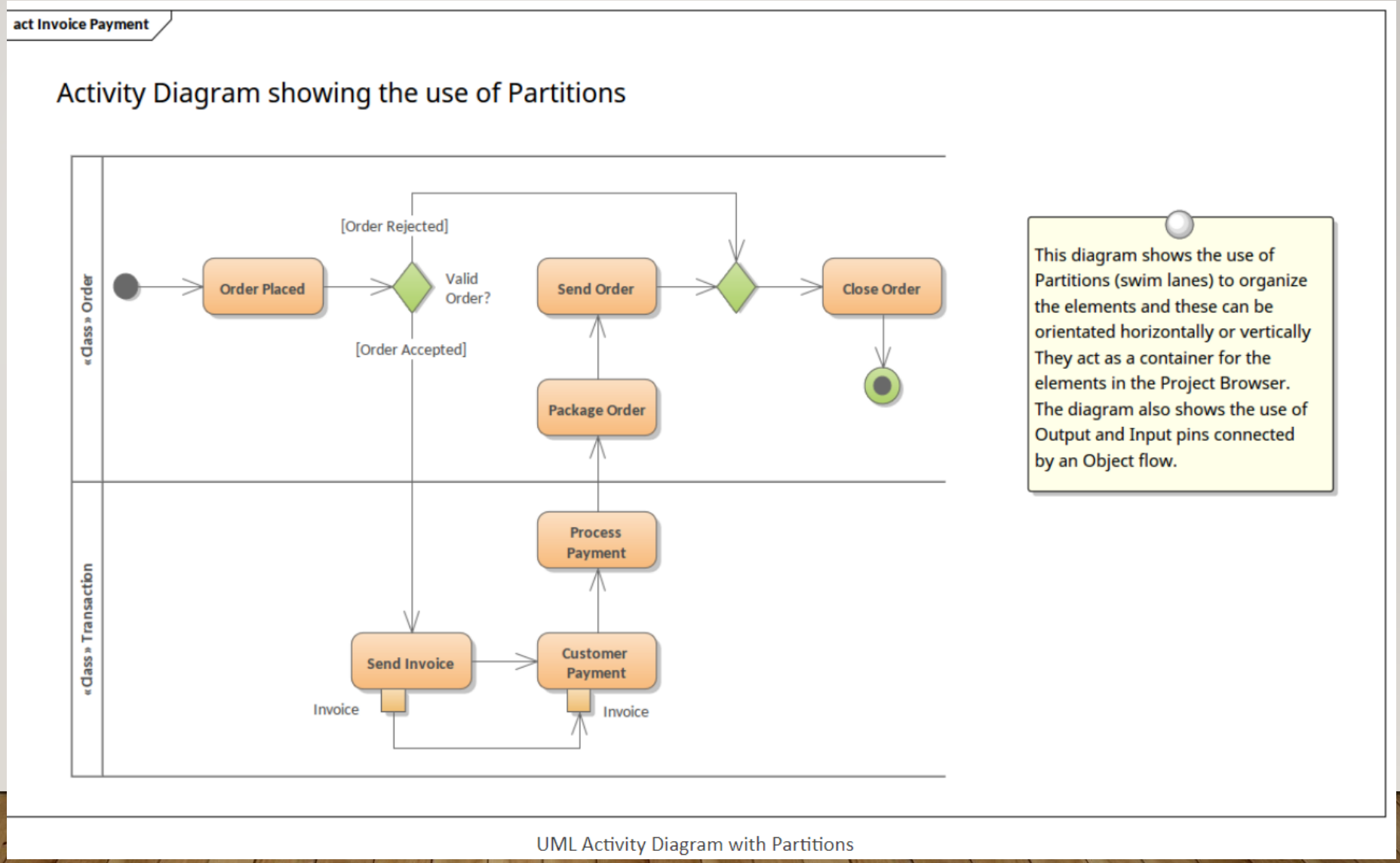
Example: In a software development process, partitions might represent different development teams, such as frontend, backend, and testing teams.
In a business process, partitions could represent various departments or roles involved in the process, such as sales, marketing, and customer support.

PARTITIONS AN EXAMPLE

Example:

The diagram shows the use of Partitions (swim lanes) to organize the elements and these can be orientated horizontally or vertically They act as a container for the elements in the **Browser window**.

The diagram also shows the use of Output and Input pins connected by an Object flow.



UML Activity Diagram with Partitions

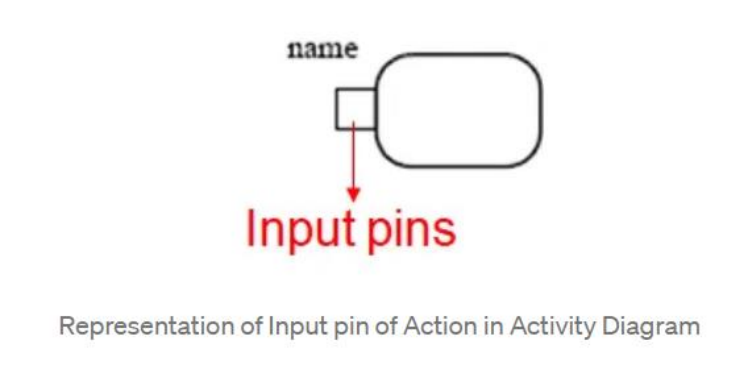
PINS IN ACTIVITY DIAGRAM

LEARN MORE

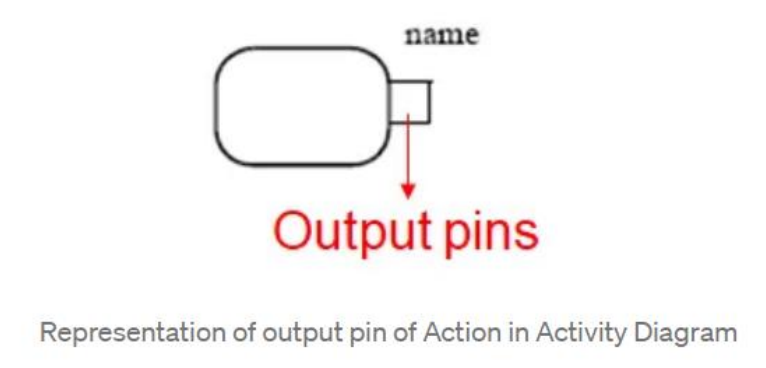
PINS

What is PIN in Activity Diagram ?

- Actions can have inputs and outputs, through the pins.
- Pins hold the inputs to actions until the action starts, and hold the output of actions before the values moves to downstream.



Representation of Input pin of Action in Activity Diagram



Representation of output pin of Action in Activity Diagram

- The name of a pin is not restricted i.e generally recalls the type of objects or data that flow through the pin.

An example for Pin representation of an action in Activity diagram

WHEN TO USE ACTIVITY DIAGRAM

LEARN MORE

WHEN TO USE ACTIVITY DIAGRAM

What you should use Activity Diagram in a Project ?

Activity diagrams are helpful in the following phases of a project:

- Before starting a project, you can create activity diagrams to model the most important workflows.
- During the requirements phase, you can create activity diagrams to illustrate the flow of events that the use cases describe.
- During the analysis and design phases, you can use activity diagrams to help define the behavior of operations.

Use activity diagrams in the following situations:

- **Analyzing a use case.**

At this stage, I'm not interested in allocating actions to objects; I just need to understand what actions need to take place and what the behavioral dependencies are. I allocate methods to objects later and show those allocations with an interaction diagram.

- **Dealing with multithreaded applications.**

I have not personally used activity diagrams for this purpose, but I have heard some good reports.

WHEN TO USE ACTIVITY DIAGRAM

Use activity diagrams in the following situations:

- **Understanding workflow:**

Even before I get into use cases, I find activity diagrams very useful for understanding a business process. I can easily draw these diagrams together with business experts to understand how a business operates and how it may change.

- **Describing a complicated sequential algorithm:**

In this case, an activity diagram is really nothing more than a UML-compliant flowchart. The usual pros and cons of flowcharts apply.

Don't use activity diagrams in the following situations:

- **Trying to see how objects collaborate:** An interaction diagram is simpler and gives you a clearer picture of collaborations.
- **Trying to see how an object behaves over its lifetime:** Use a state diagram (see Chapter 8) for that.
- **Representing complex conditional logic:** Use a truth table.

DIAGRAM

LEARN MORE

INTRODUCTION TO DIAGRAM

What is UML Diagram ?

UML, which stands for Unified Modeling Language, is a way to visually represent the architecture, design, and implementation of complex software systems.
When you're writing code, there are thousands of lines in an application, and it's difficult to keep track of the relationships and hierarchies within a software system.
UML diagrams divide that software system into components and subcomponents.

Why should you use UML diagrams?

UML is a standardized modeling language that can be used across different programming languages and development processes, so the majority of software developers will understand it and be able to apply it to their work.
UML diagrams can help engineering teams:

- Bring new team members or developers switching teams up to speed quickly.
- Navigate source code.
- Plan out new features before any programming takes place.
- Communicate with technical and non-technical audiences more easily.

INTRODUCTION TO DIAGRAM

What are the types of UML diagrams?

UML standards identify **13 types** of diagrams that are divided into two groups, defined below:

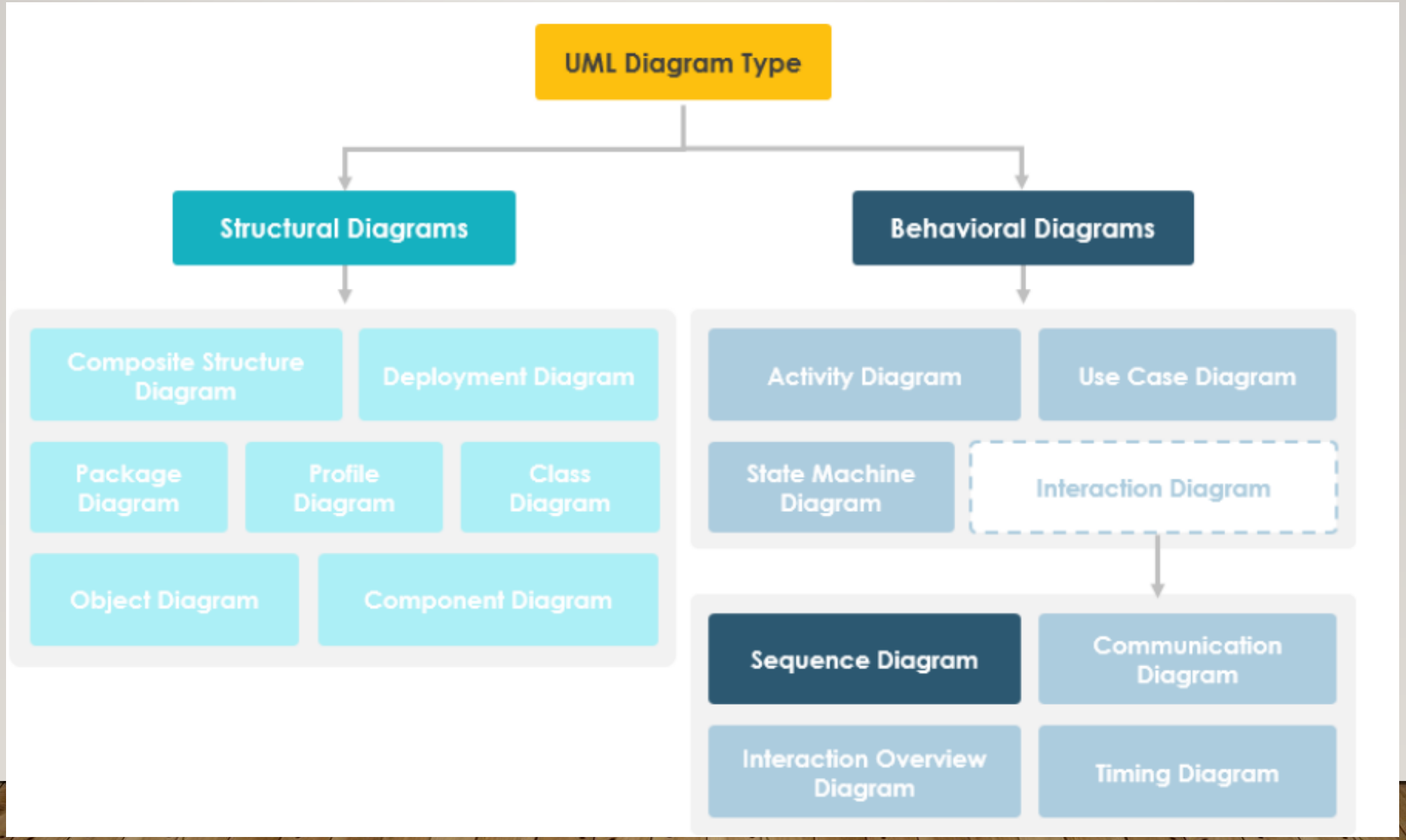
Structural UML diagrams:
Structural UML diagrams, as the name would suggest, show how the system is structured, including the classes, objects, packages, components, etc. in the system and the relationships between those elements.

Example: Class diagram, Component diagram, Deployment diagram, Composite structure diagram, Object diagram, Package diagram, Profile diagram**.**

Behavioral UML diagrams:
These UML diagrams visualize how the system behaves and interacts with itself and with users, other systems, and other entities.

Example: Timing diagram, Interaction overview diagram, Communication diagram, State diagram, Use case diagram, Sequence diagram, Activity diagram.

INTRODUCTION TO DIAGRAM



INTERACTION DIAGRAM

LEARN MORE

INTERACTION DIAGRAM

In Object-Oriented Analysis and Design (OOAD), an **Interaction Diagram** is a type of behavioral diagram that represents the interactions and relationships among objects or components within a system.
It focuses on illustrating the flow of messages or interactions between objects over time to accomplish a specific task or scenario.

There are two main types of **Interaction Diagrams** in **OOAD**:

1. **Sequence Diagram:** A sequence diagram shows the sequence of interactions and messages exchanged between objects or components over time. It depicts the order in which messages are sent and received, the lifelines of objects involved, and the activation of methods or operations. Sequence diagrams are particularly useful for modeling the dynamic behavior of a system and understanding the message flow and collaboration between objects.

2. **Communication Diagram (formerly known as Collaboration Diagram):** A communication diagram visualizes the structural relationships between objects or components and the messages exchanged among them. It represents objects as nodes connected by labeled arrows representing messages. Communication diagrams emphasize the relationships and interactions between objects, highlighting the associations and collaborations necessary to accomplish a particular task or scenario.

INTERACTION DIAGRAM

Important points about Interaction Diagrams in **OOAD:**

- Both sequence diagrams and communication diagrams are used to model the dynamic behavior of a system, but they focus on different aspects.
- Sequence diagrams emphasize the temporal ordering of messages and the sequence of method invocations, while communication diagrams emphasize the relationships and collaborations between objects.
- Interaction diagrams are valuable in the analysis and design phases of software development as they help understand and communicate the flow of interactions, identify potential issues or bottlenecks, and validate the correctness of the system's behavior.
- They provide a visual representation of how objects or components collaborate to achieve specific tasks, aiding in the design, documentation, and verification of system behavior.

The purpose of interaction diagram is:

- To capture the dynamic behavior of a system.
- To describe the message flow in the system.
- To describe the structural organization of the objects.
- To describe the interaction among objects.

SEQUENCE DIAGRAM

LEARN MORE

SEQUENCE DIAGRAM

In Object-Oriented Analysis and Design (OOAD), a sequence diagram is a type of behavioral diagram that illustrates the interactions and message exchanges between objects or components within a system. It depicts the sequence of actions and the flow of messages among objects over time, highlighting the order in which interactions occur.

A sequence diagram consists of several key elements:

1. **Objects/Participants:** Represent the entities or components (objects, classes, actors, etc.) that participate in the sequence of interactions. They are depicted as vertical lifelines or boxes.
2. **Lifelines:** Vertical lines associated with each object or participant, representing their existence over time. Lifelines show the lifespan of objects and their activation during interactions.
3. **Messages:** Arrows or lines between lifelines representing the communication or message exchanges between objects. Messages can be synchronous (blocking) or asynchronous (non-blocking). They indicate the flow of control and data between objects.
4. **Activation/Execution Occurrence:** Vertical bars or rectangles on a lifeline that show the duration of time during which an object is actively processing or executing a particular action.
5. **Combined Fragments:** Conditional or looping constructs that capture complex behavior within the sequence diagram. Examples include loops, conditionals, and parallel regions.

SEQUENCE DIAGRAM

Important Points about Sequence Diagram:

1. Sequence diagrams are commonly used to analyze and model the dynamic behavior of a system, capturing the order and timing of method calls, system responses, and collaborations between objects.
2. They are useful for visualizing and understanding the sequence of interactions in scenarios, identifying potential issues or bottlenecks, and validating the correctness of system behavior.
3. Sequence diagrams can be created at various stages of the software development lifecycle, from requirements gathering to detailed design and implementation.
4. They provide a clear and concise representation of how objects or components collaborate to accomplish a specific task or scenario, aiding in the design, communication, and documentation of system behavior.

Purpose of Sequence Diagram:

- Model high-level interaction between active objects in a system
- Model the interaction between object instances within a collaboration that realizes a use case
- Model the interaction between objects within a collaboration that realizes an operation
- Either model generic interactions (showing all possible paths through the interaction) or specific instances of a interaction (showing just one path through the interaction)

SEQUENCE DIAGRAM NOTATIONS

Sequence diagrams in Object-Oriented Analysis and Design (OOAD) use various notations to represent the elements and relationships involved in the interactions between objects. Here are the **main notations** used in sequence diagrams:

1. **Lifeline:** A vertical line or dashed line representing the lifespan of an object or participant. It indicates the existence of an object over a period of time. The lifeline is labeled with the name of the object or participant it represents.
2. **Activation/Execution Occurrence:** A vertical rectangle or bar on a lifeline indicating the period during which an object is actively processing or executing a particular action. It represents the activation of a method or operation. The activation bar is usually accompanied by a time or duration constraint.
3. **Message:** An arrow or line indicating the communication or message exchange between objects. Messages can be labeled with the name of the message, the parameters or arguments being passed, and other information. There are different types of messages:
    1. **Synchronous Message:** A solid arrow representing a blocking or synchronous message. The sender waits for a response before proceeding.
    2. **Asynchronous Message:** A dashed arrow representing a non-blocking or asynchronous message. The sender continues its execution without waiting for a response.
    3. **Return Message:** A solid arrow with a labeled dashed line indicating the return value or result of a method call.

SEQUENCE DIAGRAM NOTATIONS

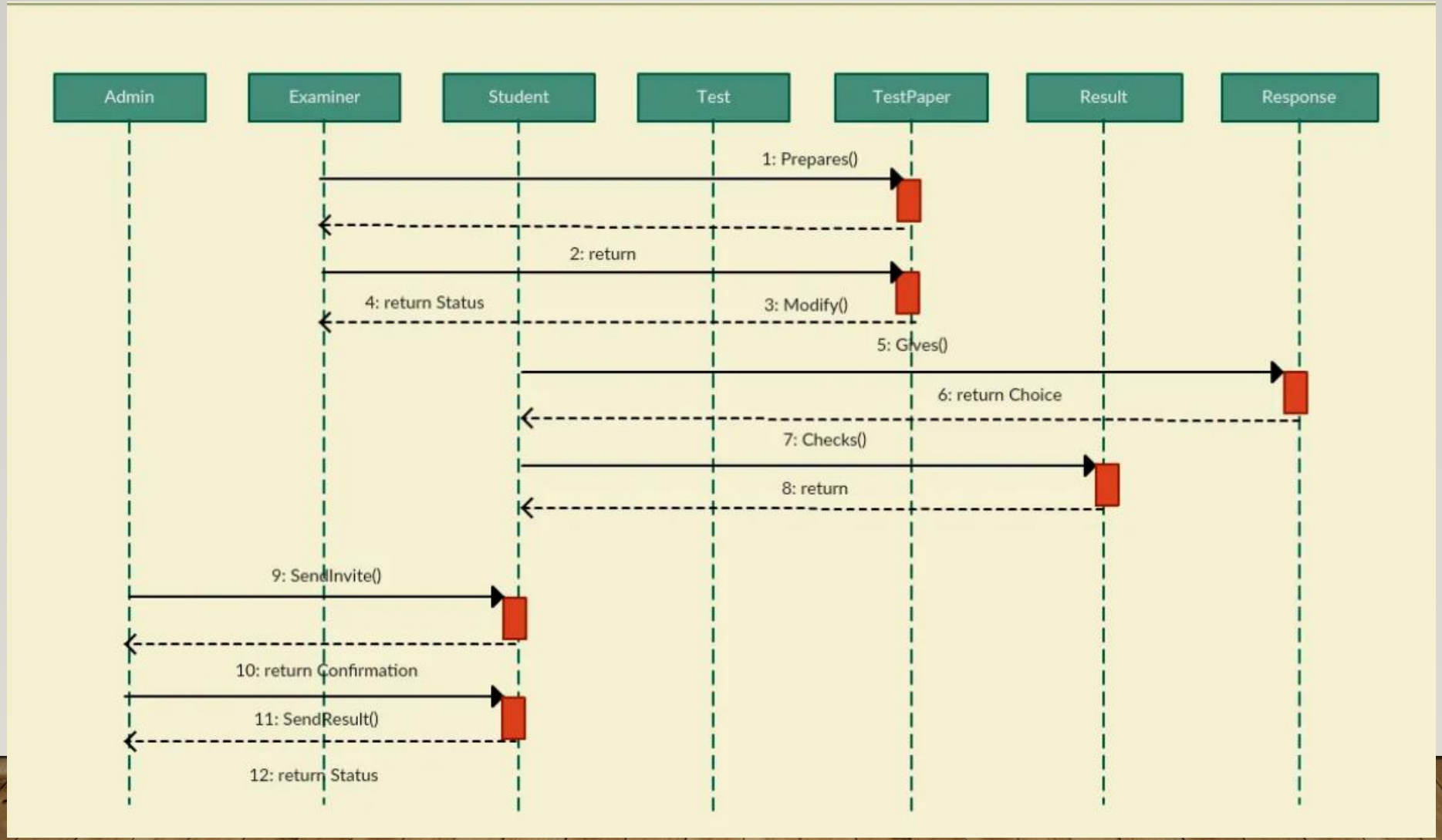
Sequence diagram Notations:

1. **Self-Message:** A message that originates from and targets the same object, representing an internal action or self-invocation within an object.
2. **Create Message:** A message that creates a new instance of an object. It is denoted by an arrow with a "+" sign.
3. **Destruction Occurrence:** An X symbol at the end of a lifeline, representing the destruction or termination of an object.
4. **Combined Fragment:** A notation used to represent conditional or looping constructs within a sequence diagram. It allows for the modeling of alternative paths, iterations, or parallel behavior.

- These notations are used together to depict the sequence of interactions, the order of method invocations, and the flow of messages between objects or participants in a system.
- They help visualize the dynamic behavior of the system and provide a clear representation of the collaboration and communication between objects during a specific scenario or task.

SEQUENCE DIAGRAM EXAMPLE

Sequence Diagram of an Online Exam System:



SEQUENCE DIAGRAM FRAMES

LEARN MORE

SEQUENCE DIAGRAM FRAMES

In sequence diagrams, **frames**, also known as **interaction frames** or **interaction operators**, are notations used to group or organize a set of related messages or interactions.
Frames provide a way to represent certain behavioral patterns or conditions within the sequence diagram. They help to structure and clarify the flow of messages and interactions between objects.
Here are some common types of frames used in sequence diagrams:

A. Combined Fragment: Combined fragments are used to represent conditional and iterative behavior within a sequence diagram. They allow for the modeling of alternative paths, repetitions, and parallel behavior.
Common types of combined fragments include:

- **Alternative (alt):** Represents a conditionally executed set of messages. It includes multiple alternative fragments, each associated with a guard condition.
- **Option (opt):** Represents an optional set of messages that may or may not be executed based on a condition.
- **Loop (loop):** Represents a set of messages that are repeatedly executed for a specified number of iterations or until a condition is met.
- **Parallel (par):** Represents parallel or concurrent execution of messages within multiple fragments.

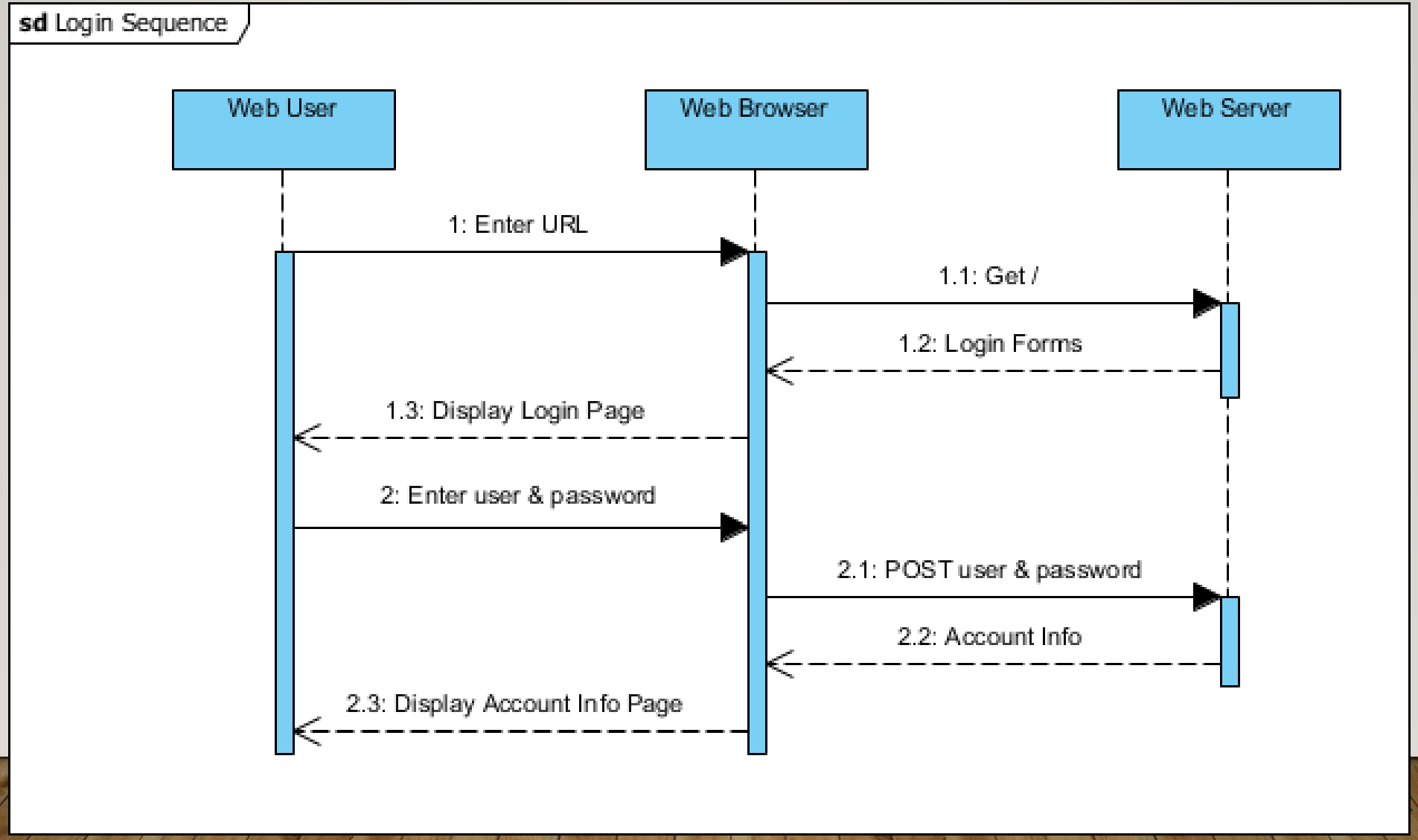
SEQUENCE DIAGRAM FRAMES

B. Interaction Operand: Interaction operands are used to represent a sequence of messages that are conditionally executed based on a guard condition. They are often used within combined fragments to define the behavior within each fragment.

C. Break (break): Represents a break or interruption in the normal flow of messages. It indicates that the sequence of interactions within the frame is interrupted or halted due to an exceptional event or condition.

- Frames help to organize complex scenarios and improve the readability of sequence diagrams by grouping related messages or interactions.
- They provide a way to express conditional, iterative, or concurrent behavior, making it easier to understand the flow of interactions between objects and the different possible paths that can be taken.
- By using frames in sequence diagrams, you can effectively represent complex behaviors and scenarios, capturing both the main flow of interactions and the alternative or exceptional paths within the system.
- A Sequence **Frame Notation** is drawn as a large rectangle with a pentagon at the top left corner. “sd” followed by the sequence name are written in the pentagon.
- **Example:** a communication message interchanges between a user, a Web browser and a Web server to perform a user login process can be described as a communication message sequence with sequence frame.

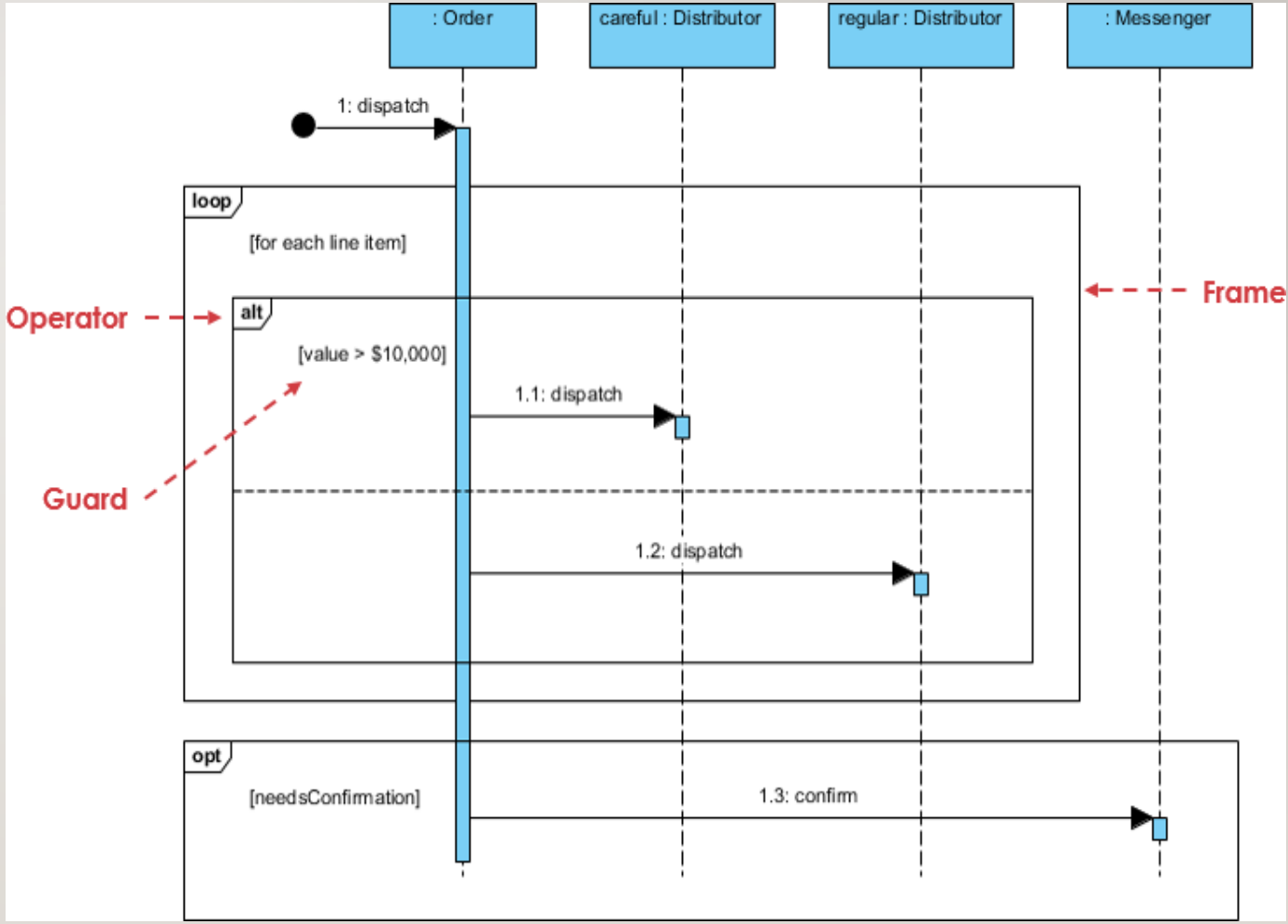
SEQUENCE DIAGRAM FRAMES



MODELING SIMPLE LOGIC AND ALGORITHMS

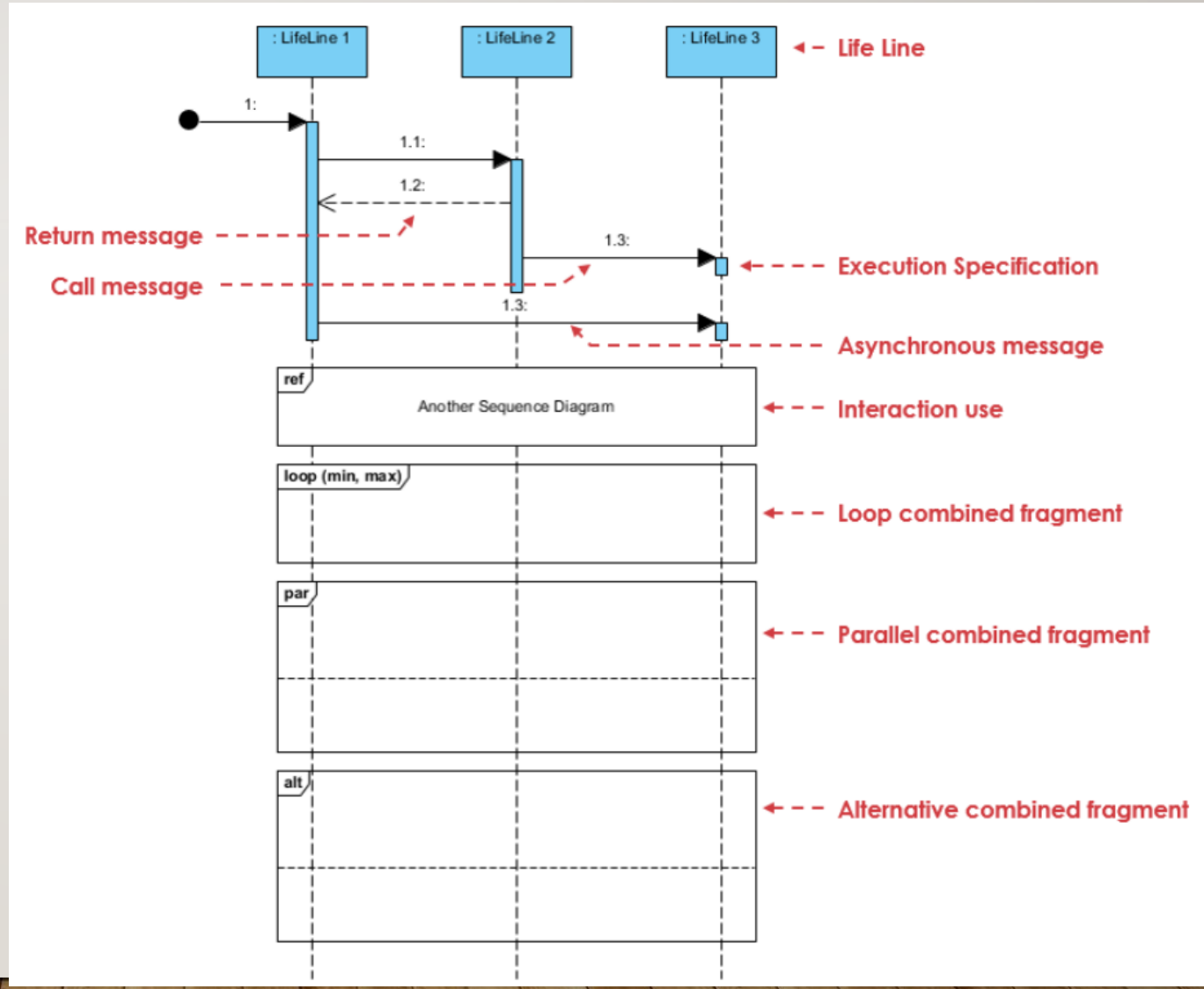
A common issue with sequence diagrams is how to show looping and conditional behavior.
The first thing to point out is that this isn't what sequence diagrams are good at.
If you want to show control structures like this, you are better off with an activity diagram or indeed with code itself.
Treat sequence diagrams as a visualization of how objects interact rather than as a way of modeling control logic.
If you still prefer to model this in a sequence diagram, here's the notation to use.
Both loop and conditional use interaction frames, which are ways of marking off a piece of a sequence diagram.

The sequence diagram example below shows a simple algorithm based on the following pseudocode:



MODELING SIMPLE LOGIC AND ALGORITHMS

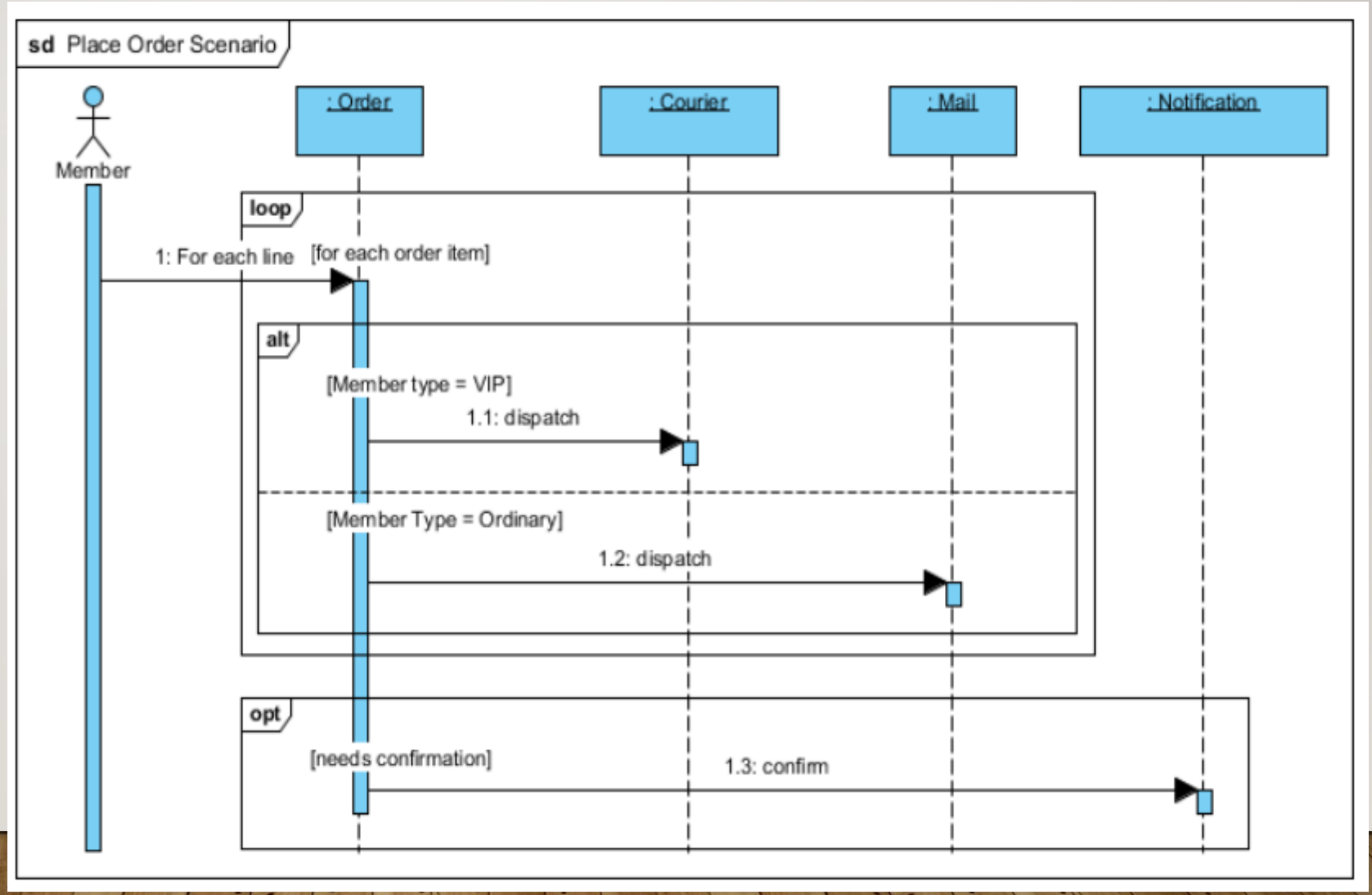
alt	Alternative multiple fragments: only the one whose condition is true will execute.
opt	Optional: the fragment executes only if the supplied condition is true. Equivalent to an alt only with one trace.
par	Parallel: each fragment is run in parallel.
loop	Loop: the fragment may execute multiple times, and the guard indicates the basis of iteration.
region	Critical region: the fragment can have only one thread executing it at once.
neg	Negative: the fragment shows an invalid interaction.
ref	Reference: refers to an interaction defined in another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and return value.
sd	Sequence diagram: used to surround an entire sequence diagram.



SEQUENCE FRAGMENT EXAMPLE- PLACE ORDER SCENARIO

Example:

- A member of a ship who would like to place an order online.
- The item ordered will be sent to the member either send by courier or by ordinary mail depending on the member status (VIP, Ordinary membership).
- **Optionally**, the shop will send the member a confirmation notice if the member opted for the notification option in the order.



INTRODUCTION TO OBJECT STATE

LEARN MORE

INTRODUCTION TO OBJECT STATE & UML

In UML (Unified Modeling Language), **object state** refers to the condition or status of an object at a particular point in time. It represents the values of an object's attributes or properties and reflects its behavior and characteristics. Object state is an essential aspect of modeling the dynamic behavior of a system and understanding how objects interact and respond to events and stimuli.
In UML, **object state** is typically represented using state diagrams, which are a type of behavioral diagram. State diagrams visually depict the different states an object can be in and the transitions between those states. They provide a graphical representation of how an object's behavior changes based on internal or external events.

The key elements used to represent object state in UML state diagrams are:
1.States: States represent the different conditions or modes that an object can be in. Each state represents a distinct configuration of the object's attributes and behavior. States are depicted as rounded rectangles and are labeled with their names.

2.Transitions: Transitions represent the change from one state to another in response to an event or trigger. They indicate how an object transitions from one state to another based on certain conditions or actions. Transitions are represented as directed arrows connecting states and are labeled with the triggering event or condition.

INTRODUCTION TO OBJECT STATE & UML

The key elements used to represent object state in UML state diagrams are:

1.Events: Events represent the occurrences or stimuli that trigger a state transition. They can be internal events, such as the completion of an action or a timer expiration, or external events, such as user input or messages received from other objects.

2.Actions: Actions represent the activities or behaviors associated with a state or transition. They specify the operations or activities performed when entering or exiting a state or during a state transition.
By using state diagrams to represent object state, you can capture and visualize the various states an object can be in, the events that trigger state transitions, and the actions performed during those transitions.
This helps in understanding and modeling the behavior of objects within a system, including how they respond to events, interact with other objects, and exhibit different behaviors based on their current state.

STATE MACHINE DIAGRAM

LEARN MORE

STATE MACHINE DIAGRAM & NOTATIONS

What is State Machine Diagram ?

A state machine diagram, also known as a statechart diagram, is a type of behavioral diagram in the Unified Modeling Language (UML). It is used to model the behavior of a system or an individual object as a finite-state machine. State machine diagrams depict the different states that an object or system can be in and the transitions between those states in response to events or conditions.

Here are the key elements used in a state machine diagram:

1.States: States represent the distinct conditions or modes that an object or system can be in. Each state represents a specific configuration of attributes and behavior. States are depicted as rounded rectangles and are labeled with their names.

2.Transitions: Transitions represent the movements or changes from one state to another. They depict the conditions or events that trigger the transition and the actions or activities performed during the transition. Transitions are represented as arrows and are labeled with the triggering event or condition.

3.Initial State: The initial state represents the starting point of the state machine diagram. It indicates the state in which an object or system begins its behavior.

4.Final State: The final state represents the end point or termination of the state machine diagram. It indicates the state in which an object or system reaches the completion of its behavior.

STATE MACHINE DIAGRAM

What is State Machine Diagram ?

5. Events: Events represent the occurrences or stimuli that trigger state transitions. They can be external events, such as user actions or messages received, or internal events, such as the completion of an action or a timer expiration.

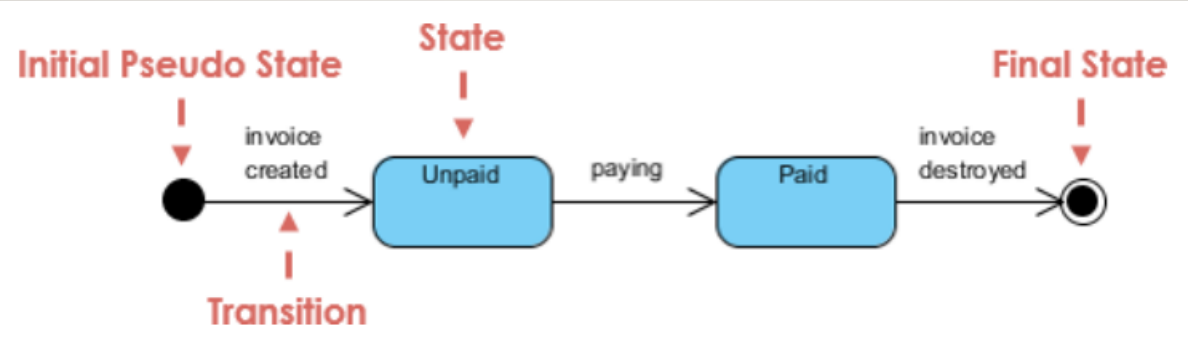
6. Actions: Actions represent the activities or behaviors associated with a state or transition. They specify the operations or activities performed when entering or exiting a state or during a state transition.

7. Guards: Guards are conditional expressions associated with transitions that determine if the transition is taken or not. They represent conditions that must be satisfied for the transition to occur. Guards are written near the transitions, typically inside square brackets or curly braces.

- State machine diagrams are particularly useful for modeling and understanding the behavior of complex systems, where the behavior is influenced by various states and transitions. They provide a visual representation of how an object or system responds to events and changes its state over time.
- State machine diagrams help in analyzing and designing systems with dynamic behavior, identifying potential issues or conflicts, and ensuring the completeness and correctness of the system's behavior.

STATE MACHINE DIAGRAM EXAMPLE

Simple State Machine Diagram Notation:



Entry and Exit Actions:

Entry and Exit actions specified in the state. It must be true for every entry / exit occurrence. If not, then you must use actions on the individual transition arcs:

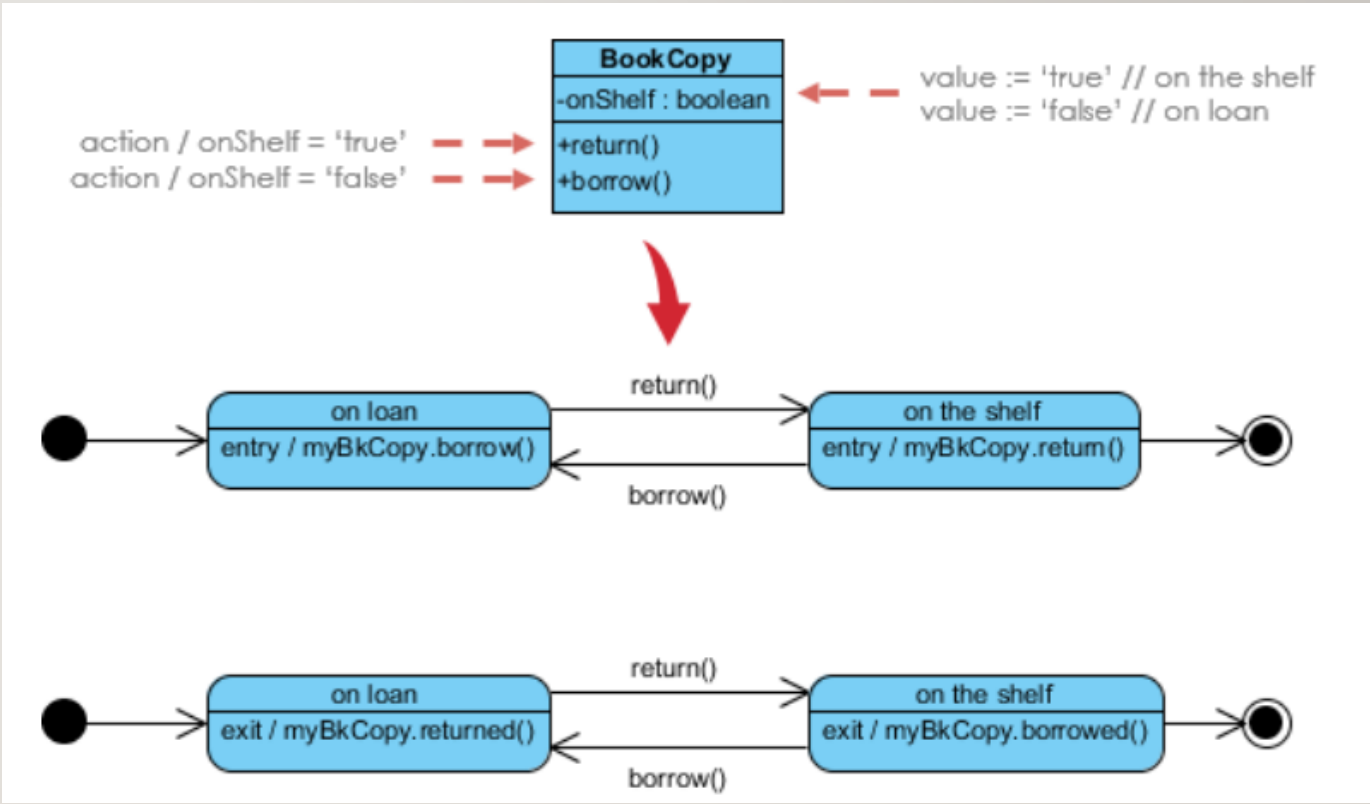
- **Entry Action** executed on entry into state with the **notation: Entry / action**
- **Exit Action** executed on exit from state with the **notation: Exit / action**

STATE MACHINE DIAGRAM EXAMPLE

Example - Entry / Exit Action (Check Book Status):
This example illustrates a state machine diagram derived from a Class - "BookCopy":

Note:

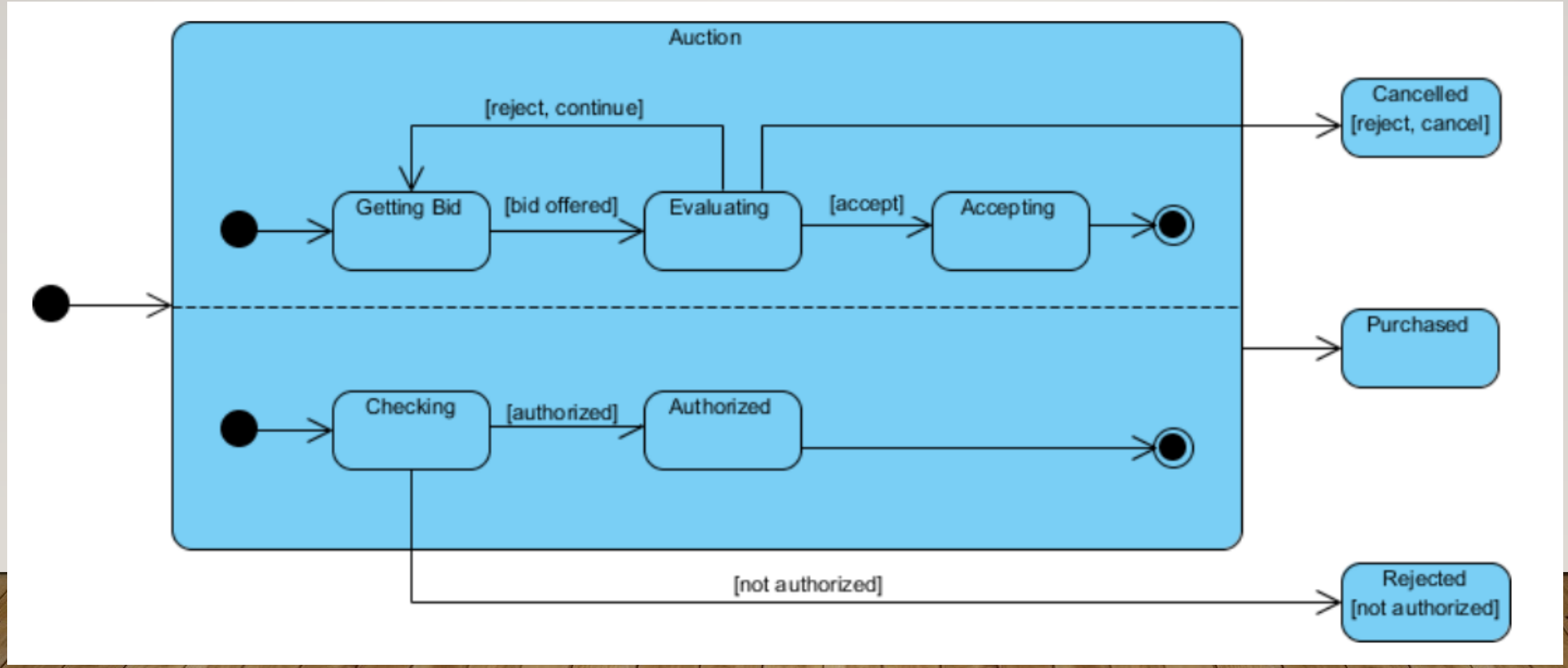
1. This state machine diagram shows the state of an object myBkCopy from a BookCopy class
2. **Entry action :** any action that is marked as linked to the entry action is executed whenever the given state is entered via a transition
3. **Exit action :** any action that is marked as linked to the exit action is executed whenever the state is left via a transition



STATE MACHINE DIAGRAM EXAMPLE

Example - Concurrent State Machine Diagram Example - Auction Process :

- In this example, the state machine first entering the Auction requires a fork at the start into two separate start threads.
- Each substate has an exit state to mark the end of the thread. Unless there is an abnormal exit (Canceled or Rejected), the exit from the composite state occurs when both substates have exited.



STUDIO SHODWE

THANK YOU