

Lab 2: Sequence Diagram Revision

Why Sequence Diagram?

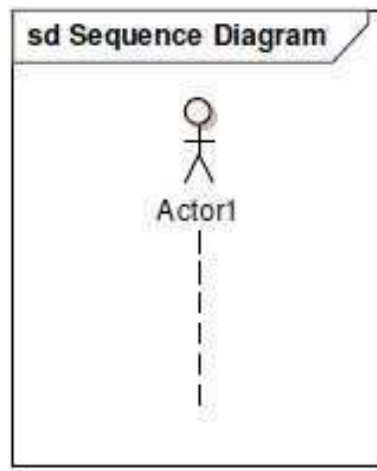
UML sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artifact for dynamic modeling, which focuses on identifying the behavior within your system. One of the primary uses of sequence diagrams is in the transition from requirements expressed as use cases to the next and more formal level of refinement. Use cases are often refined into one or more sequence diagrams.

The main purpose of a sequence diagram is to define event sequences that result in some desired outcome. The focus is less on messages themselves and more on the order in which messages occur; nevertheless, most sequence diagrams will communicate what messages are sent between a system's objects as well as the order in which they occur. The diagram shows this information along the horizontal and vertical dimensions: the vertical dimension shows, top down, the time sequence of messages/calls as they occur, and the horizontal dimension shows, left to right, the object instances that the messages are sent to.

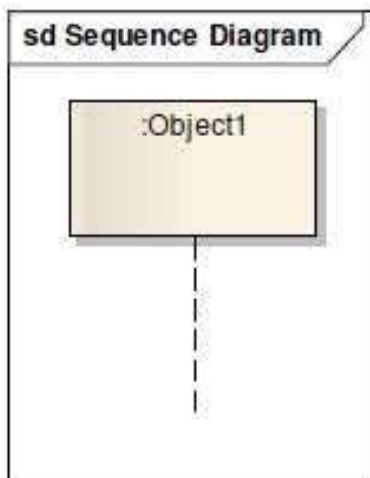
Diagram Notations

Now that we have a basic idea of how the sequence diagram works and why we use it, let's see how we can draw a one. Below we will discuss the different elements of a Sequence Diagram and learn about the role of each one of them:




1. **Actor:** The actor is the person who interacts with the system or the subcomponents of the system





2. **Object:** An object is an instance of a class; objects can communicate with each other and interact with the actor in order to fulfill what is required. The vertical dotted line under the object represents its lifetime.



- 3. Messages:** The table below shows the different type of messages that can be sent to objects and their meanings.

Message	Description
	Synchronous: A synchronous message between active objects indicates wait semantics; the sender waits for the message to be handled before it continues. This typically shows a method call.
	Asynchronous: With an asynchronous flow of control, there is no explicit return message to the caller. An asynchronous message between objects indicates no-wait semantics; the sender does not wait for the message before it continues. This allows objects to execute concurrently.
	Reply: This shows the return message from another message.

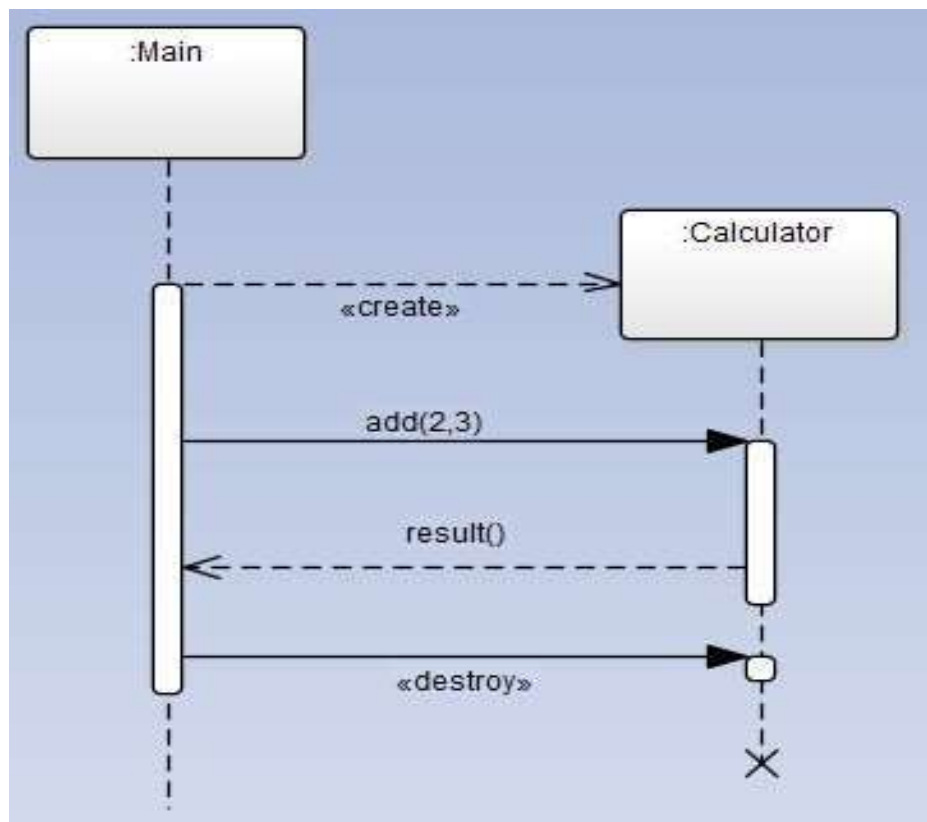
Message	Description
	Create: This message results in the creation of a new object. The message could call a constructor for a class if you are working with Java, for example.
	When an element is destroyed during an interaction, the communication that destroys the element is shown with its arrowhead to the elements lifeline where the destruction is marked with a large X symbol.

Real Life Example:

In the example below, you can see the main method inside a Java class called Main creating a new object from the class Calculator which has a method called **add** that takes two numbers and returns the result back to the caller.

```
public class Main {  
    public static void main(String[] args) {  
        Calculator calc = new Calculator();  
        calc.add(2, 3);  
    }  
}
```

Below is a sequence diagram that represents the code shown above:



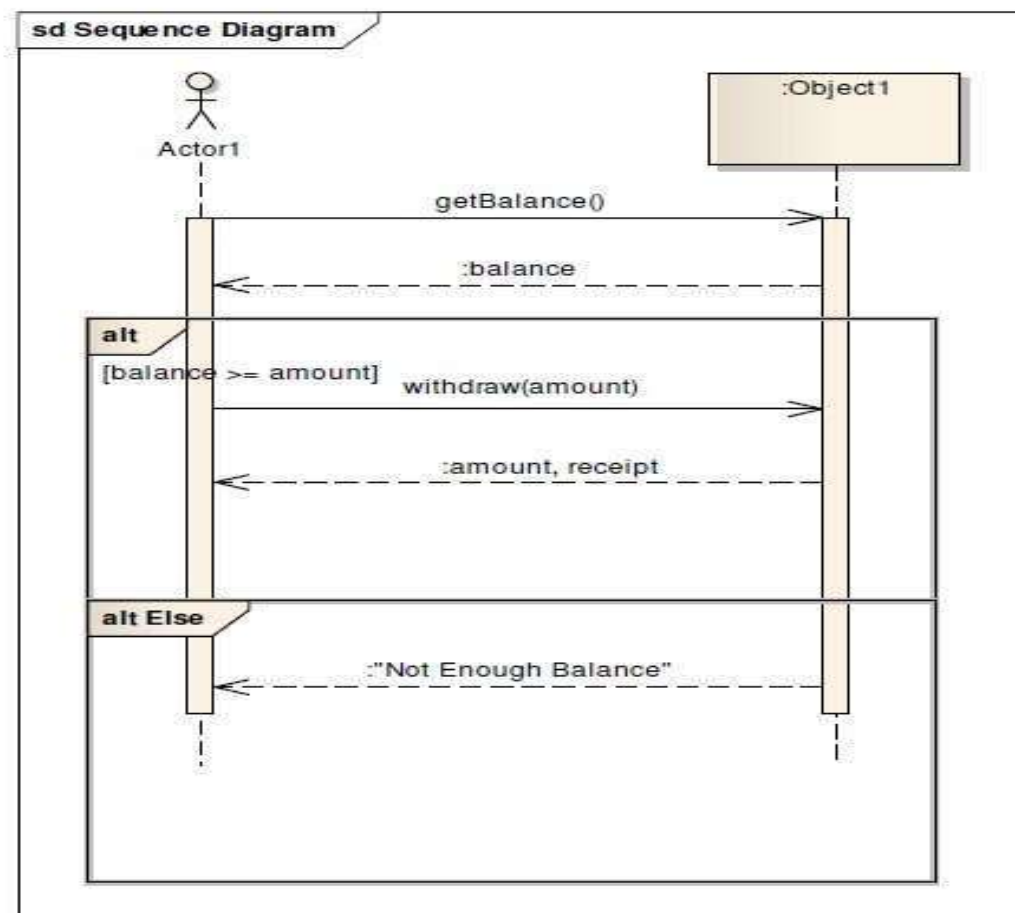
Sequence Fragments

Sequence fragments make it easier to create and maintain accurate sequence diagrams.

A sequence fragment is represented as a box, called a combined fragment, which encloses portion of the interactions within a sequence diagram. The fragment operator (in the top left corner) indicates the type of fragment. Most popular Fragment types: loop, alt, opt.

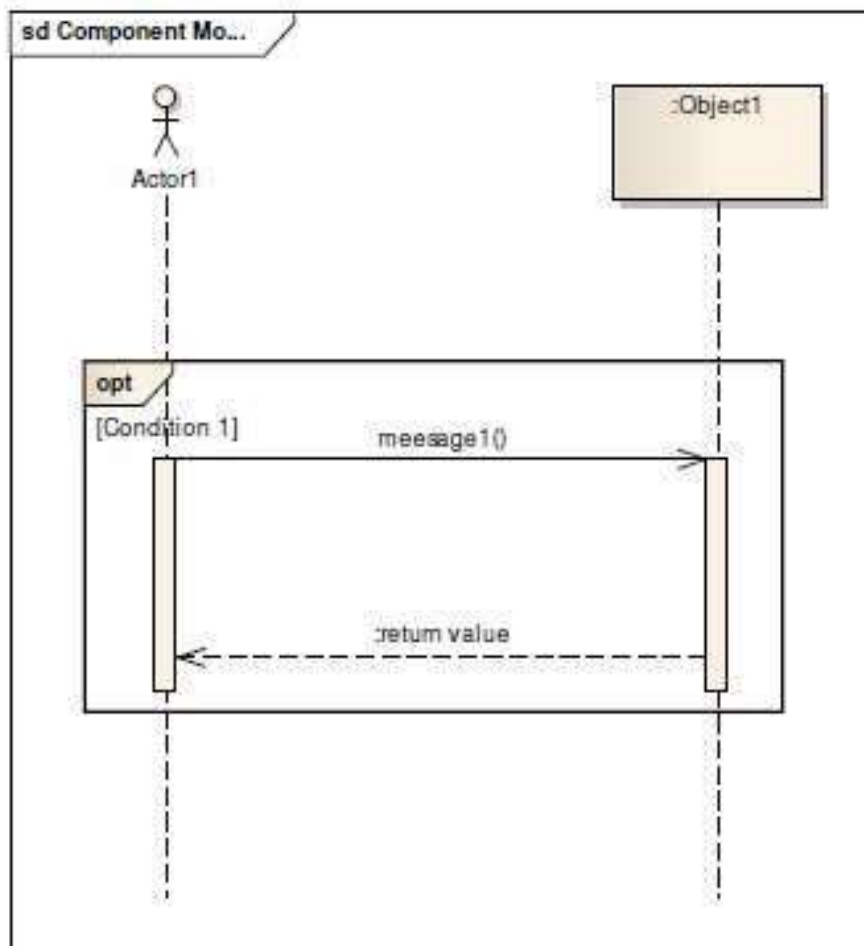
Alt

Alt indicates that one of some possible sequences (within the scope of the alt) will be executed based on a guard (condition), alt represents an if –else structure.



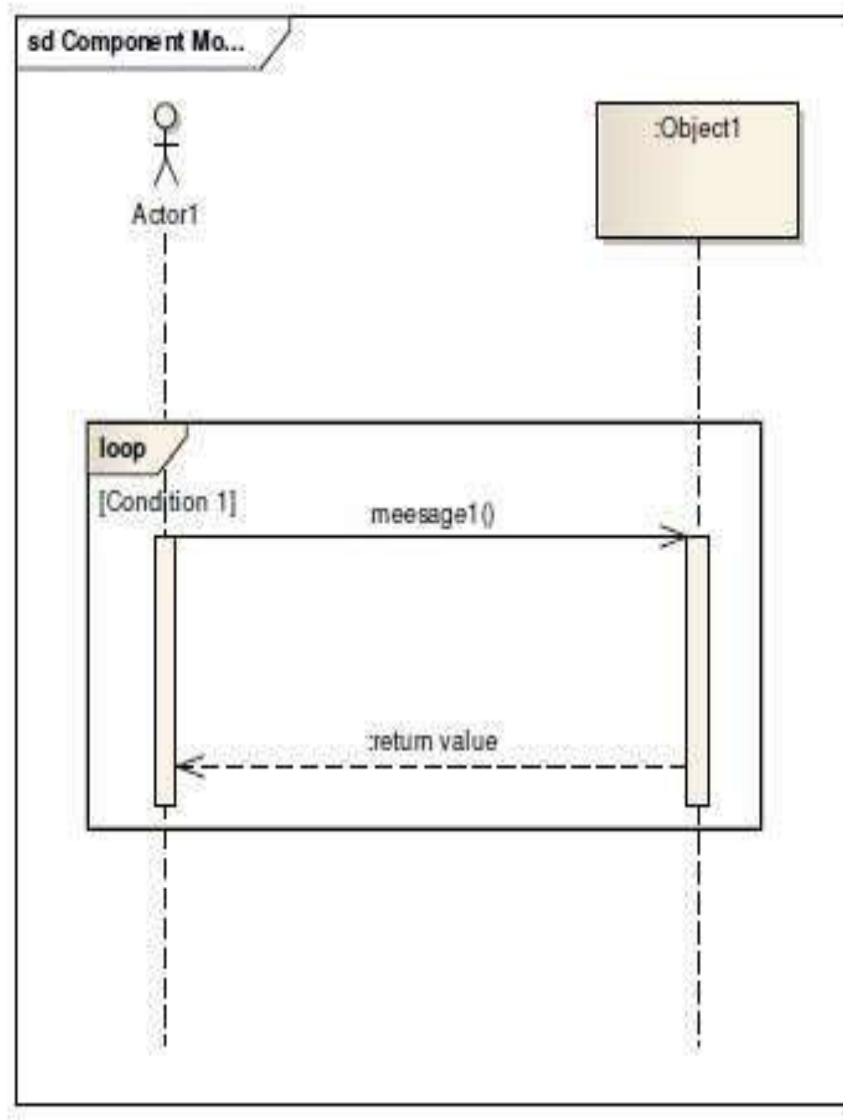
Opt

The Opt in a sequence diagram indicates an optional part of the sequence that can either be performed or not based on a guard (condition).

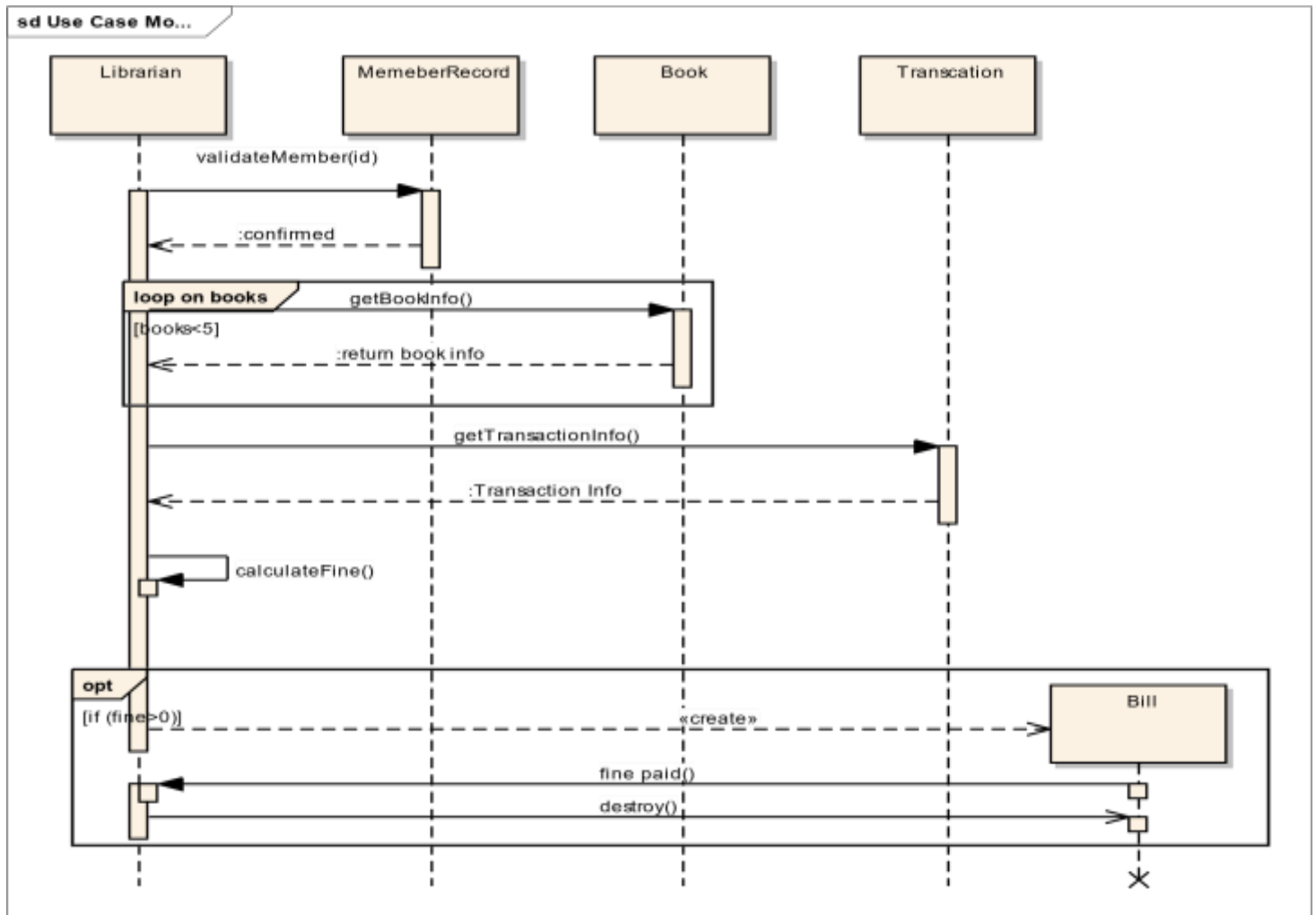


Loop

A loop indicates that a certain sequence will be repeated as long as the specified guard (condition) holds.



Example 1: Sequence Diagram for (return book use case) from Library Management Case Study



System Sequence Diagram

A system sequence diagram (SSD) is a sequence diagram that shows, for a particular scenario of a use case, the events that external actors generate, their order, and possible inter-system events. It deals with the system as a one huge object. Below is a system sequence diagram for **withdraw money** use case from the ATM case study.

