Sequence Diagram

This sequence diagram tutorial is to help you understand sequence diagrams better; to explain everything you need to know, from how to draw a sequence diagram to the common mistakes you should avoid when drawing one.

There are 3 types of Interaction diagrams; Sequence diagrams, communication diagrams, and timing diagrams. These diagrams are used to illustrate interactions between parts within a system. Among the three, sequence diagrams are preferred by both developers and readers alike for their simplicity.

What is a Sequence Diagram?

Sequence diagrams, commonly used by developers, model the interactions between objects in a single use case. They illustrate how the different parts of a system interact with each other to carry out a function, and the order in which the interactions occur when a particular use case is executed.

In simpler words, a sequence diagram shows different parts of a system work in a ‘sequence’ to get something done.

Benefits of using an interaction diagram

Interaction diagrams can be implemented in a number of scenarios to provide a unique set of information. They can be used to:

- Model a system as a time-ordered sequence of events.
- Reverse- or forward-engineer a system or process.
- Organize the structure of various interactive events.
- Simply convey the behavior of messages and lifelines within a system.
- Identify possible connections between lifeline elements.

Types of interaction diagrams in UML

Interaction diagrams are divided into four main types of diagrams:

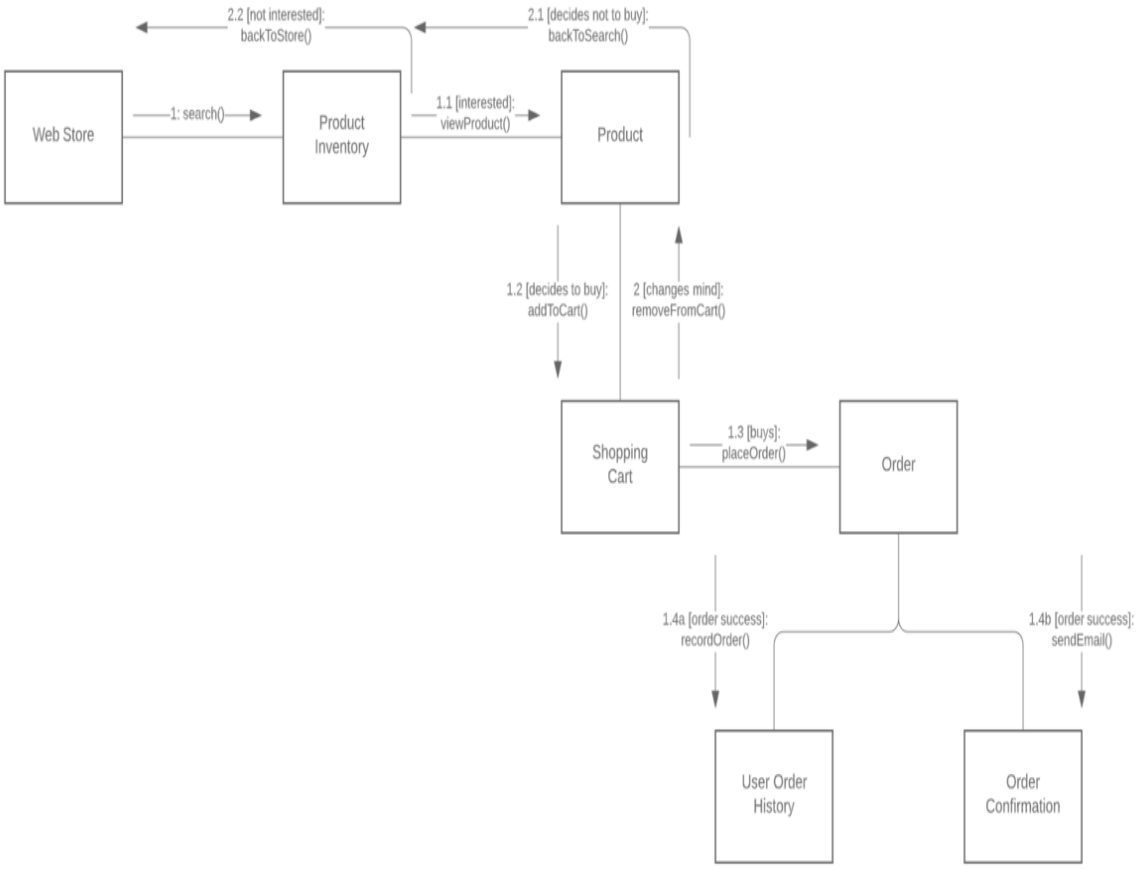
Communication diagram (or collaboration diagram)

- Sequence diagram
- Timing diagram
- Interaction overview diagram

Each type of diagram focuses on a different aspect of a system's behavior or structure. Take a look below for more information on the basics of each diagram and how you can benefit from them.

Communication diagram (or collaboration diagram)

In UML, communication diagrams depict the relationships and interactions among various software objects. They emphasize the structural aspects of an interaction diagram, focusing on object architecture rather than the flow of messages.



A communication diagram provides the following benefits:

- They emphasize how lifelines connect.
- They focus on elements within a system rather than message flow.
- They provide an added emphasis on organization over timing.

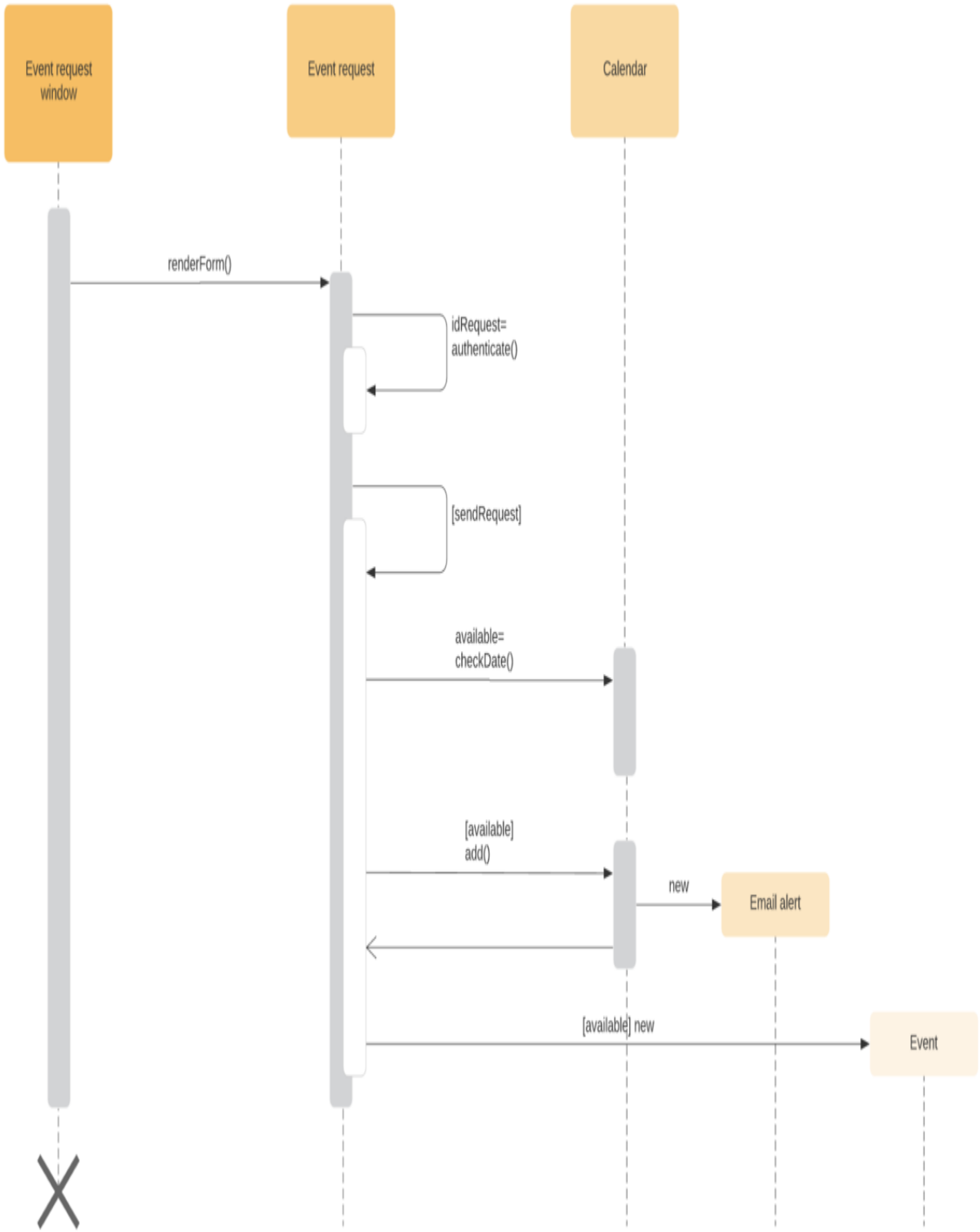
Communication diagrams can also have these possible downsides:

- They can become very complex.
- They make it difficult to explore specific objects within a system.
- They can be time-consuming to create.

Sequence diagram

Another option for depicting interactions is using sequence diagrams. These diagrams revolve around five main events:

- Order placement
- Payment
- Order confirmation
- Order preparation
- Order serving



If the sequence of events changes, it can cause delays, or the system may crash. It's important to select the notation that matches the particular sequence within your diagram.

A sequence diagram provides the following benefits:

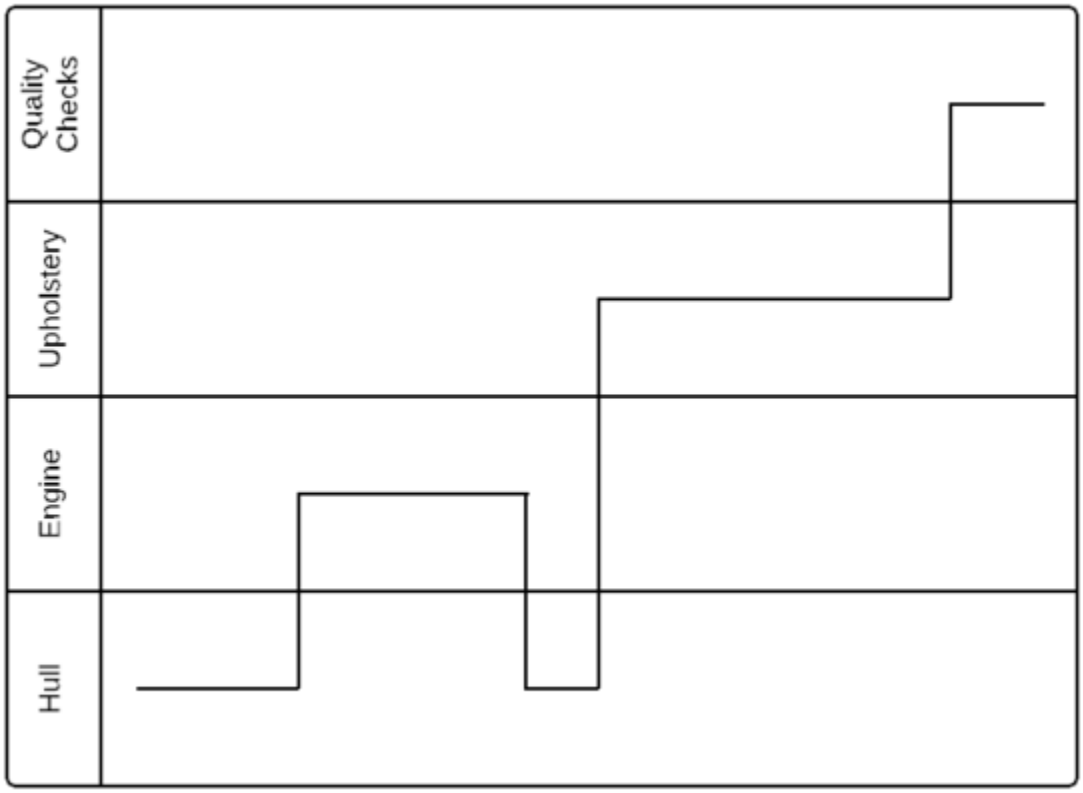
- They're easy to maintain and generate.
- They're easy to update according to changes in a system.
- They allow for reverse and forward engineering.

Sequence diagrams can also have these possible downsides:

- They can become complex, with too many lifelines and varied notations.
- They're easy to produce incorrectly and depend on your sequence being entered correctly.

Timing diagram

Another diagram option can be to use timing diagrams. These are visuals used to depict the state of a lifeline at any instance in time, denoting the changes in an object from one form to another. Waveforms are used within timing diagrams to visualize the flow within the software program at various instances of time.



A timing diagram offers the following benefits:

- They allow for forward and reverse engineering.
- They can represent the state of an object at an exact instance in time.
- They can keep track of any and all changes within a system.

You should also consider these potential downsides of using a timing diagram:

- They can be difficult to understand.
- They can be hard to maintain over time.

Interaction overview diagram

The interaction overview diagram provides a high-level view of an interaction model. The diagram acts as an overview of the flow of control from interaction to interaction, as well as the flow of activity from diagram to diagram.

A timing diagram offers the following benefits:

- They provide an uncomplicated view of the activity within a model.
- They offer a high degree of navigability between diagrams.
- They allow the use of most annotations within an activity diagram, along with additional elements for added clarity.

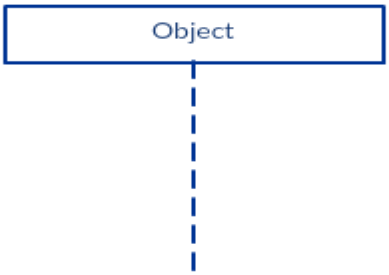
Although interaction diagrams are fairly intuitive, they do require branching and interactions to follow certain behaviors, which can be restrictive.

Sequence Diagram Notations

A sequence diagram is structured in such a way that it represents a timeline which begins at the top and descends gradually to mark the sequence of interactions. Each object has a column and the messages exchanged between them are represented by arrows.

A Quick Overview of the Various Parts of a Sequence Diagram

Lifeline Notation



A sequence diagram is made up of several of these lifeline notations that should be arranged horizontally across the top of the diagram. No two lifeline notations should overlap each other. They represent the different objects or parts that interact with each other in the system during the sequence.

A lifeline notation with an actor element symbol is used when the particular sequence diagram is owned by a use case.

A lifeline with an entity element represents system data. For example, in a customer service application, the Customer entity would manage all data related to a customer.

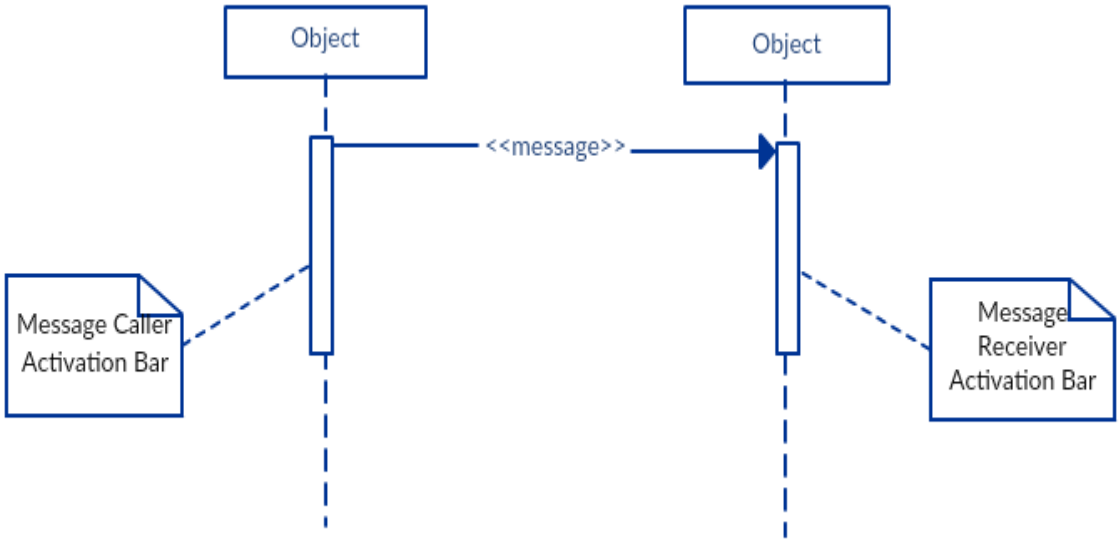
A lifeline with a boundary element indicates a system boundary/ software element in a system; for example, user interface screens, database gateways or menus that users interact with, are boundaries.

And a lifeline with a control element indicates a controlling entity or manager. It organizes and schedules the interactions between the boundaries and entities and serves as the mediator between them.

Activation Bars

Activation bar is the box placed on the lifeline. It is used to indicate that an object is active (or instantiated) during an interaction between two objects. The length of the rectangle indicates the duration of the objects staying active.

In a sequence diagram, an interaction between two objects occurs when one object sends a message to another. The use of the activation bar on the lifelines of the Message Caller (the object that sends the message) and the Message Receiver (the object that receives the message) indicates that both are active/is instantiated during the exchange of the message.



Message Arrows

An arrow from the Message Caller to the Message Receiver specifies a message in a sequence diagram. A message can flow in any direction; from left to right, right to left or back to the Message Caller itself.

While you can describe the message being sent from one object to the other on the arrow, with different arrowheads you can indicate the type of message being sent or received.

The message arrow comes with a description, which is known as a message signature, on it. The format for this message signature is below.

All parts except the message_name are optional.

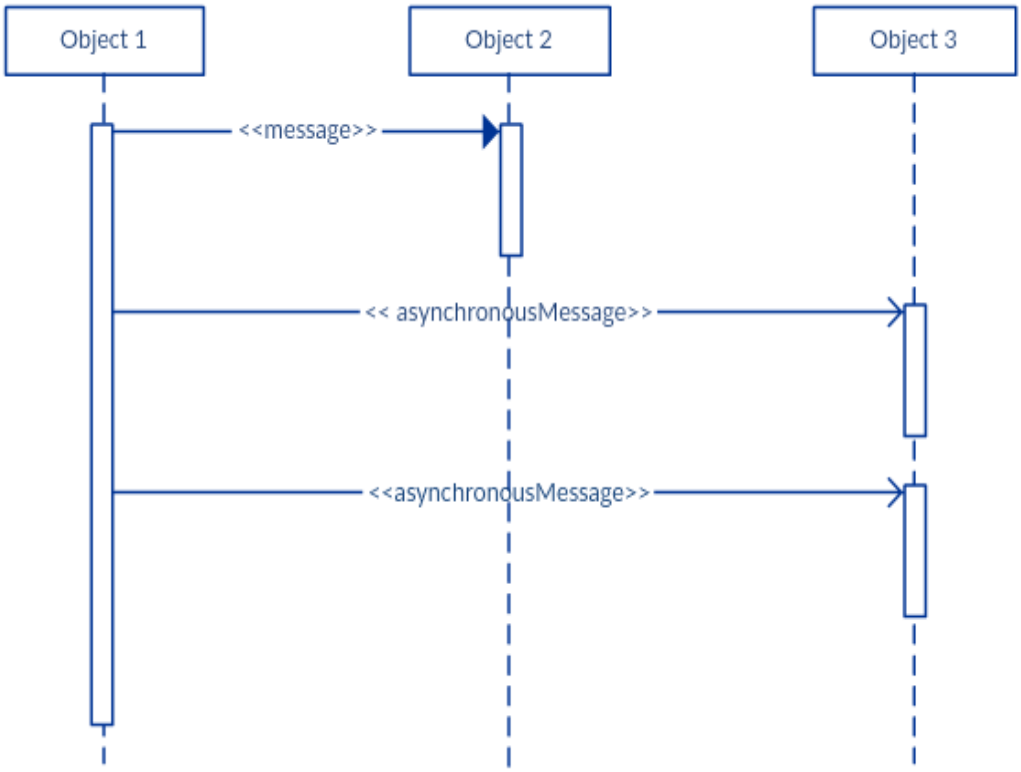
attribute = message_name (arguments): return_type

- *Synchronous message*

As shown in the activation bars example, a synchronous message is used when the sender waits for the receiver to process the message and return before carrying on with another message. The arrowhead used to indicate this type of message is a solid one, like the one below.

- *Asynchronous message*

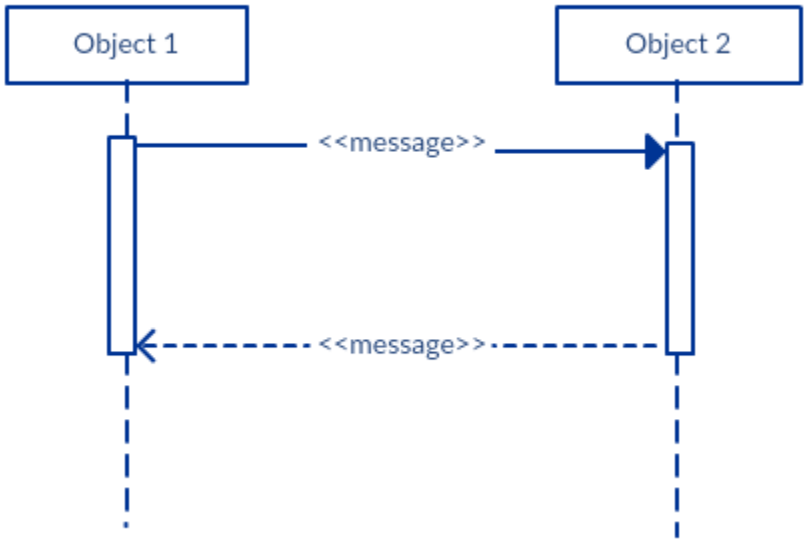
An asynchronous message is used when the message caller does not wait for the receiver to process the message and return before sending other messages to other objects within the system. The arrowhead used to show this type of message is a line arrow like shown in the example below.



- *Return message*

A return message is used to indicate that the message receiver is done processing the message and is returning control over to the message caller. Return messages are optional notation pieces, for an activation bar that is triggered by a synchronous message always implies a return message.

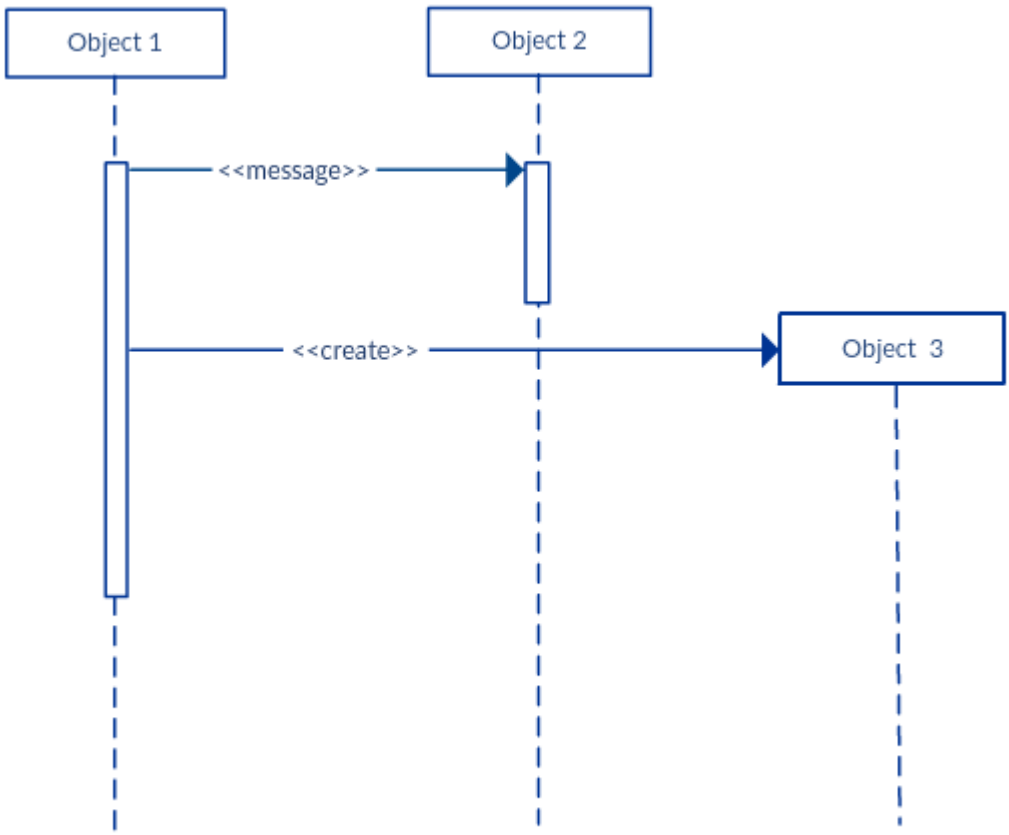
Tip: You can avoid cluttering up your diagrams by minimizing the use of return messages since the return value can be specified in the initial message arrow itself.



- *Participant creation message*

Objects do not necessarily live for the entire duration of the sequence of events. Objects or participants can be created according to the message that is being sent.

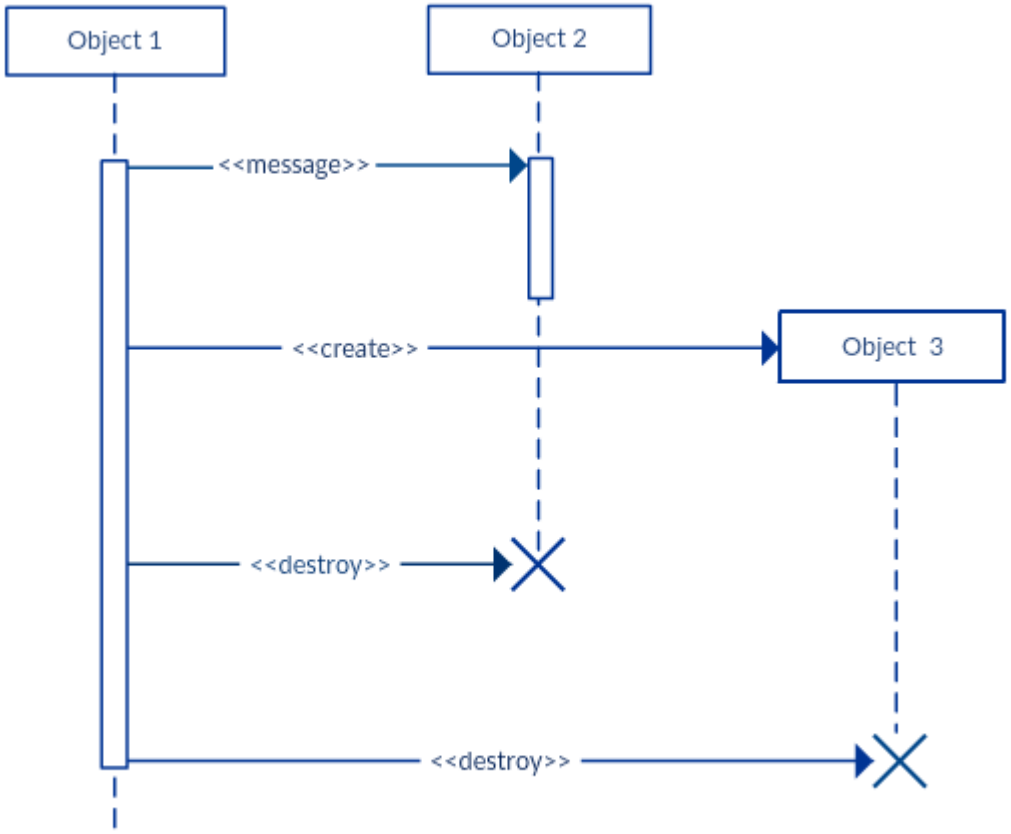
The dropped participant box notation can be used when you need to show that the particular participant did not exist until the create call was sent. If the created participant does something immediately after its creation, you should add an activation box right below the participant box.



- *Participant destruction message*

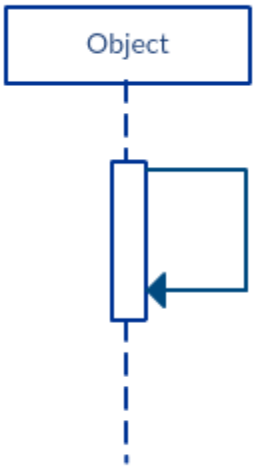
Likewise, participants when no longer needed can also be deleted from a sequence diagram. This is done by adding an ‘X’ at the end of the lifeline of

the said participant.



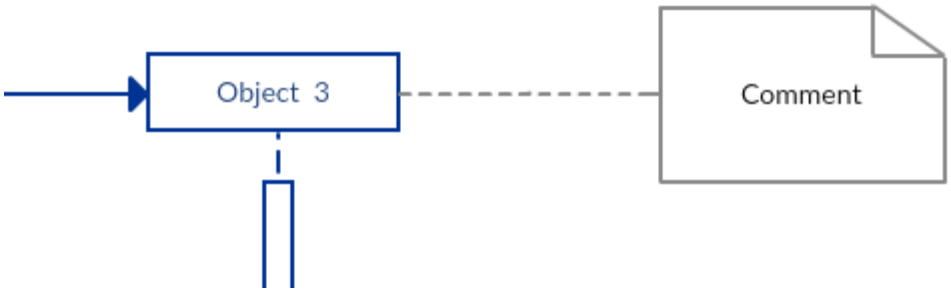
- *Reflexive message*

When an object sends a message to itself, it is called a reflexive message. It is indicated with a message arrow that starts and ends at the same lifeline as shown in the example below.



Comment

UML diagrams generally permit the annotation of comments in all UML diagram types. The comment object is a rectangle with a folded-over corner as shown below. The comment can be linked to the related object with a dashed line.



Note: View Sequence Diagram Best Practices to learn about sequence fragments.

Sequence Diagram Best Practices

- **Manage complex interactions with sequence fragments**

A sequence fragment is represented as a box that frames a section of interactions between objects (as shown in the examples below) in a sequence diagram.

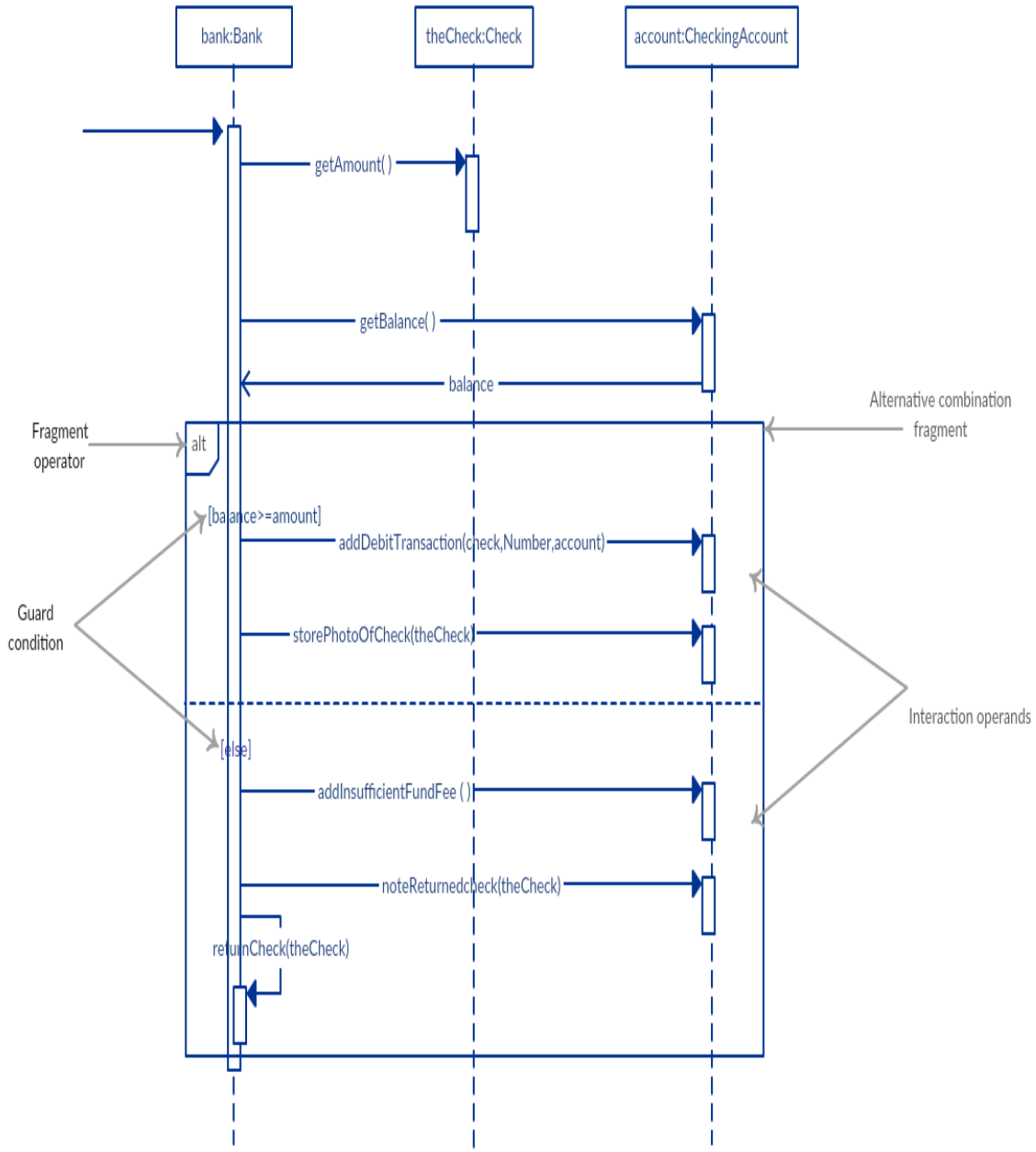
It is used to show complex interactions such as alternative flows and loops in a more structured way. On the top left corner of the fragment sits an operator. This – the fragment operator – specifies what sort of a fragment it is.

Alternatives

The alternative combination fragment is used when a choice needs to be made between two or more message sequences. It models the “if then else” logic.

The alternative fragment is represented by a large rectangle or a frame; it is specified by mentioning ‘alt’ inside the frame’s name box (a.k.a. fragment operator).

To show two or more alternatives, the larger rectangle is then divided into what is called interaction operands using a dashed line, as shown in the sequence diagram example above. Each operand has a guard to test against and it is placed at the top left corner of the operand.



Options

The option combination fragment is used to indicate a sequence that will only occur under a certain condition, otherwise, the sequence won't occur. It models the "if then" statement.

Similar to the alternative fragment, the option fragment is also represented with a rectangular frame where 'opt' is placed inside the name box.

Unlike the alternative fragment, an option fragment is not divided into two or more operands. Option's guard is placed at the top left corner.

(Find an example sequence diagram with an option fragment in the Sequence Diagram Templates and Examples section).

Loops

Loop fragment is used to represent a repetitive sequence. Place the words 'loop' in the name box and the guard condition near the top left corner of the frame.

In addition to the Boolean test, the guard in a loop fragment can have two other special conditions tested against. These are minimum iterations (written as *minint = [the number]* and maximum iterations (written as maxint = [the number]).

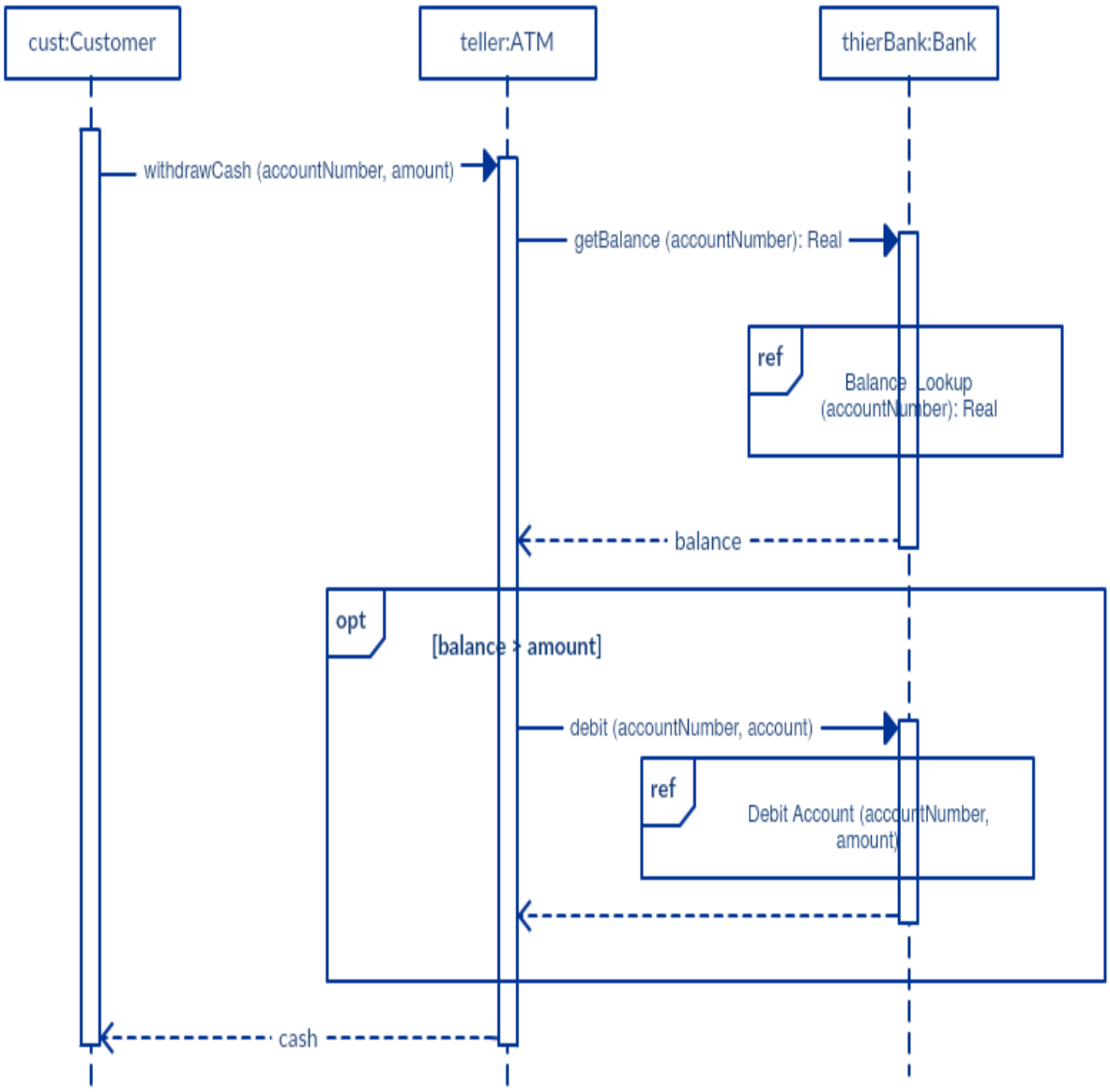
If it is a minimum iterations guard, the loop must execute not less than the number mentioned, and if it is a maximum iterations guard, the loop mustn't execute more than the number indicated.

(Find an example of a loop fragment below in the sequence diagram templates and example section)

Reference Fragment

You can use the ref fragment to manage the size of large sequence diagrams. It allows you to reuse part of one sequence diagram in another, or in other words, you can reference part of a diagram in another diagram using the ref fragment.

To specify the reference fragment, you have to mention 'ref' in the name box of the frame and the name of the sequence diagram that is being referred to inside the frame.



For more sequence fragments refer to Beyond the Basics of Sequence Diagrams: Part 1, Part 2 and Part 3.

- **Draw smaller sequence diagrams that capture the essence of the use case**

Instead of cluttering your sequence diagram with several objects and groups of messages that will confuse the reader, draw a few smaller sequence diagrams that aptly explain what your system does. Make sure that the diagram fits on a single page and leaves space for explanatory notes too.

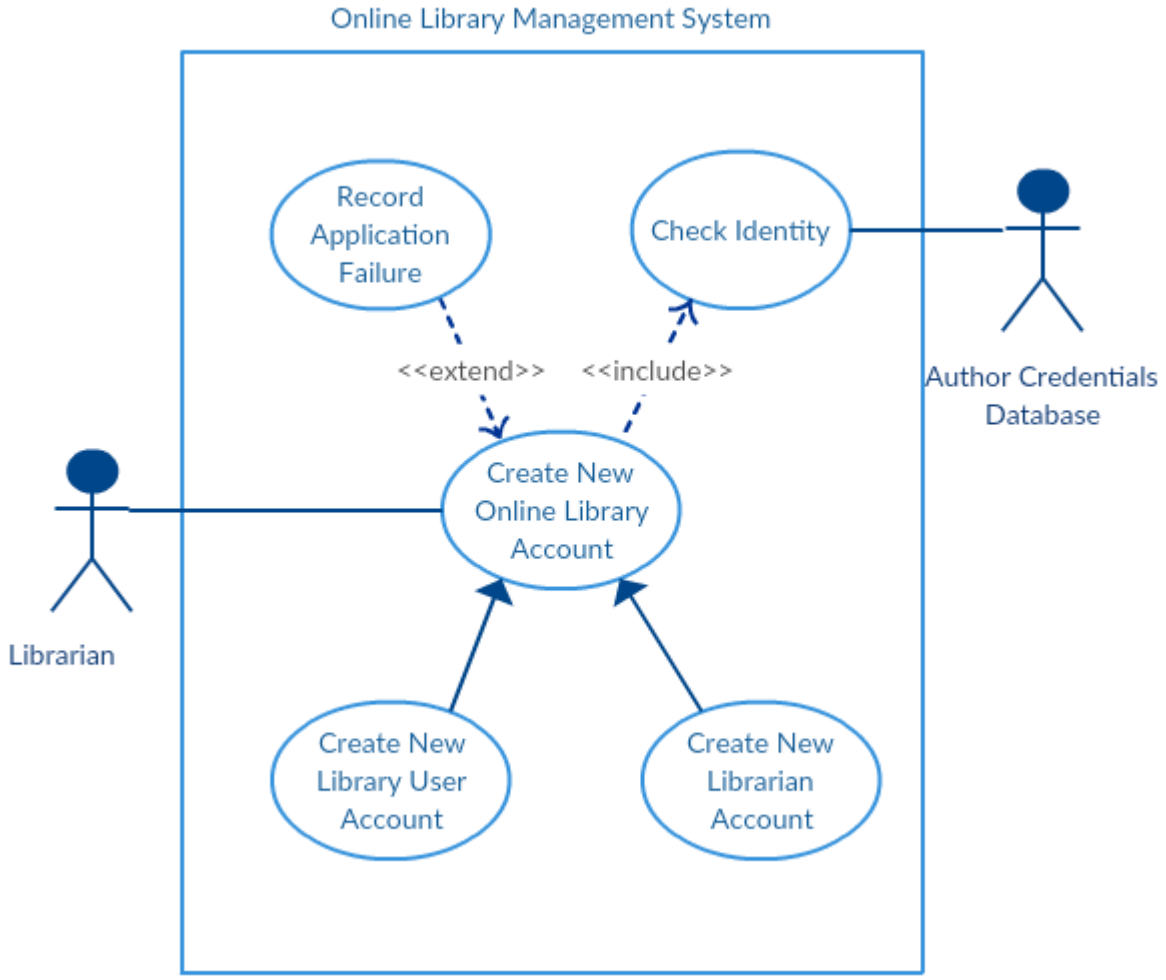
Also instead of drawing dozens of sequence diagrams, find out what is common among the scenarios and focus on that. And if the code is

expressive and can stand on its own, there’s no need to draw a sequence diagram in the first place.

How to Draw a Sequence Diagram

A sequence diagram represents the scenario or flow of events in one single use case. The message flow of the sequence diagram is based on the narrative of the particular use case.

Then, before you start drawing the sequence diagram or decide what interactions should be included in it, you need to draw the use case diagram and ready a comprehensive description of what the particular use case does.



From the above use case diagram example of ‘Create New Online Library Account’, we will focus on the use case named ‘Create New User Account’ to draw our sequence diagram example.

Before drawing the sequence diagram, it’s necessary to identify the objects or actors that would be involved in creating a new user account. These would be;

- Librarian
- Online Library Management system
- User credentials database
- Email system

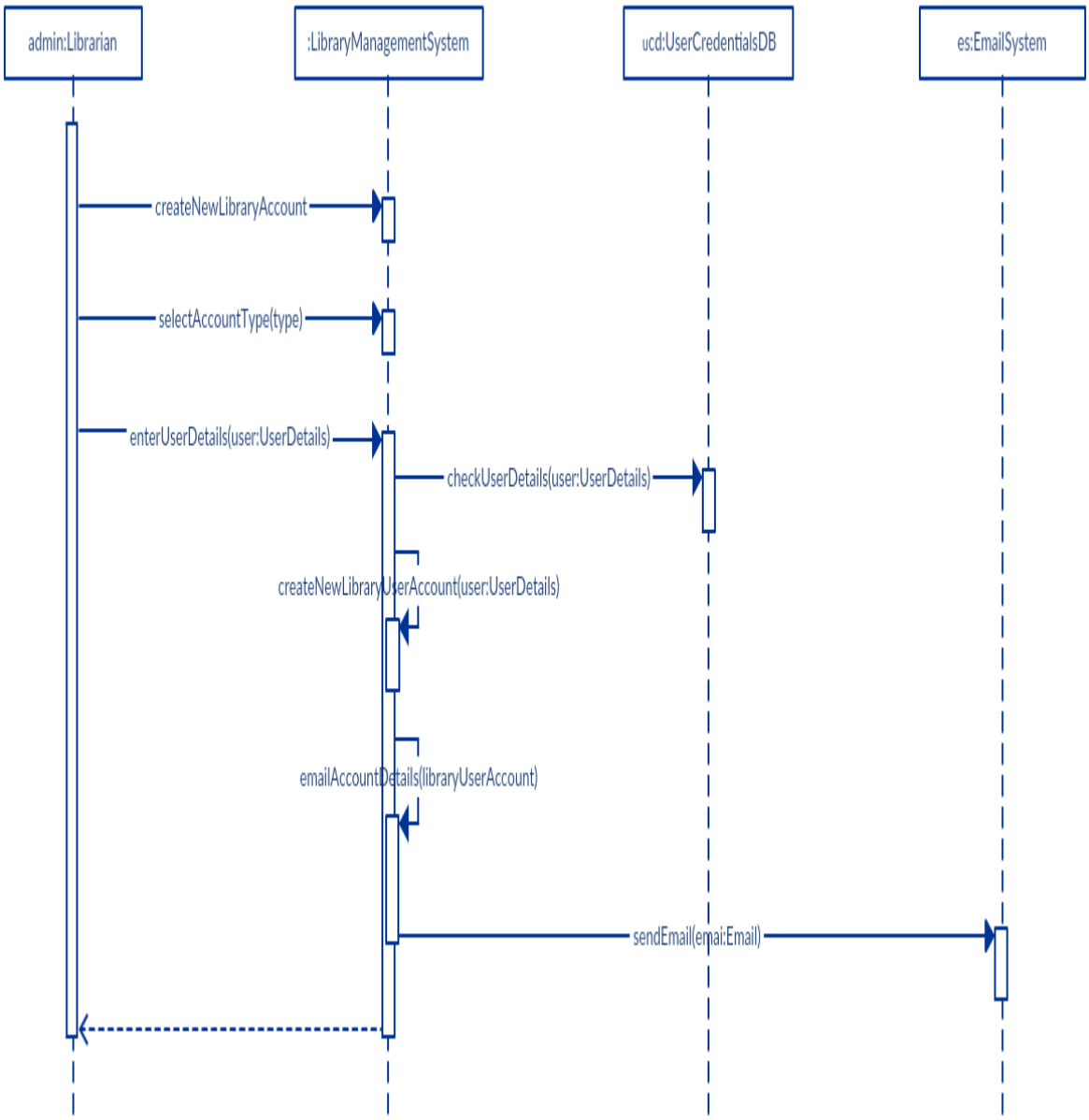
Once you identify the objects, it is then important to write a detailed description on what the use case does. From this description, you can easily figure out the interactions (that should go in the sequence diagram) that would occur between the objects above, once the use case is executed.

Here are the steps that occur in the use case named ‘Create New Library User Account’.

- The librarian request the system to create a new online library account
- The librarian then selects the library user account type
- The librarian enters the user’s details
- The user’s details are checked using the user Credentials Database
- The new library user account is created
- A summary of the of the new account’s details are then emailed to the user

From each of these steps, you can easily specify what messages should be exchanged between the objects in the sequence diagram. Once it’s clear, you can go ahead and start drawing the sequence diagram.

The sequence diagram below shows how the objects in the online library management system interact with each other to perform the function ‘Create New Library User Account’.



Sequence Diagram Common Mistakes

When drawing sequence diagrams, designers tend to make these common mistakes. By avoiding these mistakes you can ensure the quality of your diagram.

- Adding too much detail. This clutters up the diagram and makes it difficult to read.
- Obsolete and out of date sequence diagrams that are irrelevant when compared to the interfaces, actual architectures etc. of the system. Don’t forget to replace them or modify them.

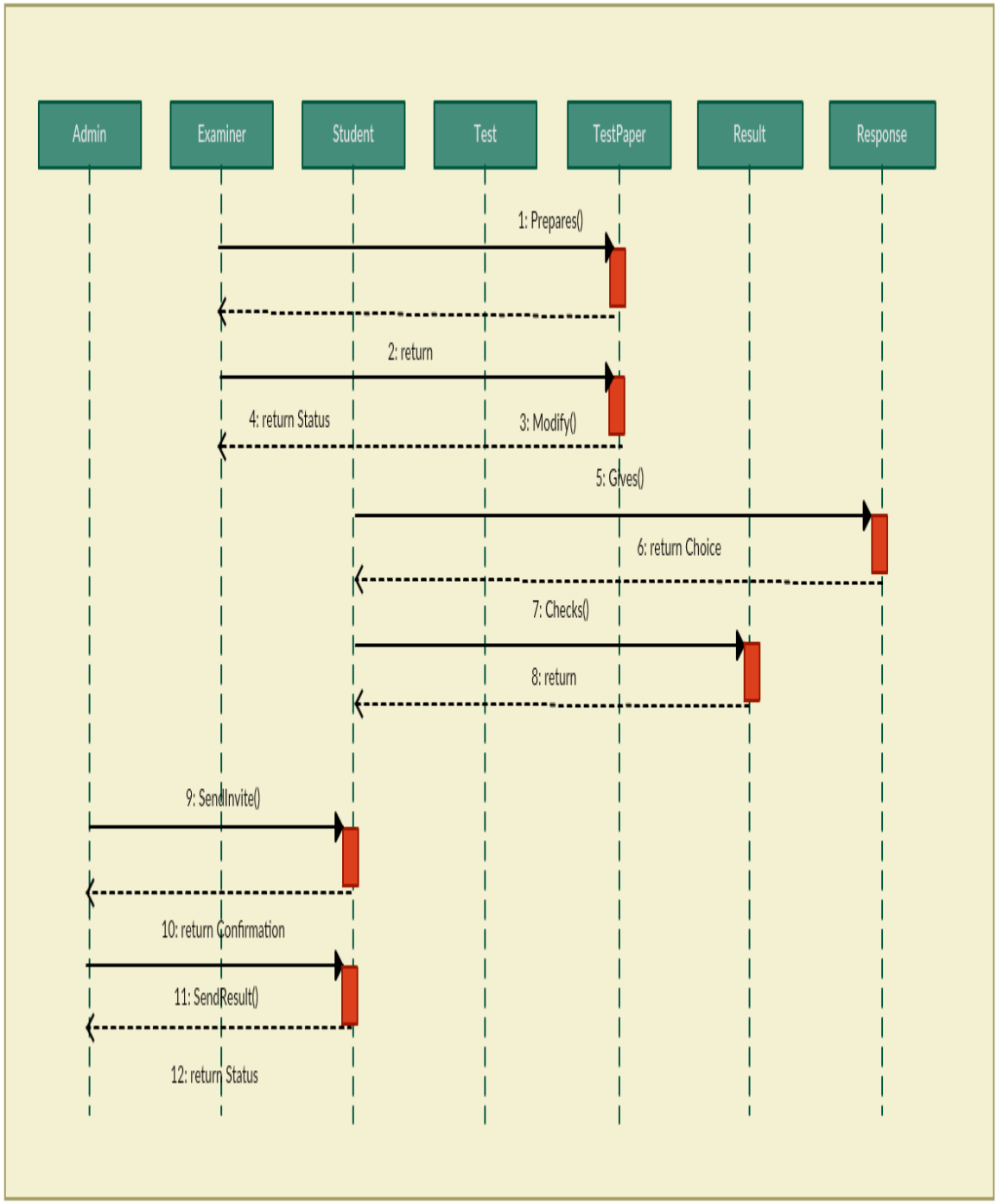
- Leaving no blank space between the use case text and the message arrow; this makes it difficult for anyone to read the diagram.
- Not considering the origins of message arrows carefully.

See these common mistakes explained in detail in Sequence Diagram Guide: Common Mistakes to Avoid When Drawing Sequence Diagrams.

Sequence Diagram Examples and Templates

Following are a few sequence diagram examples and templates that are drawn using Creately. Create sequence diagrams online using Creately's online tool. Click on the template to open it in the editor.

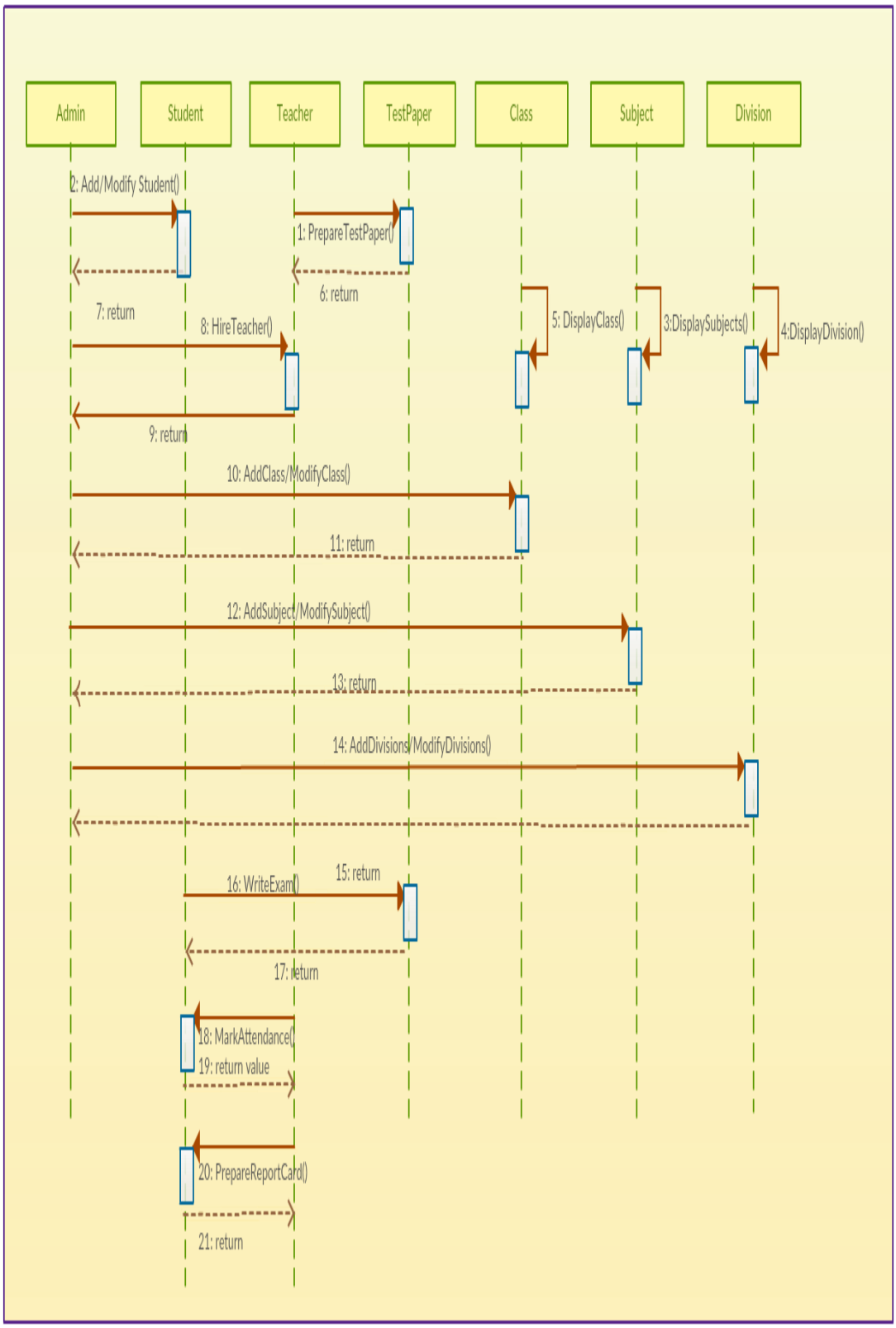
Sequence Diagram of an Online Exam System



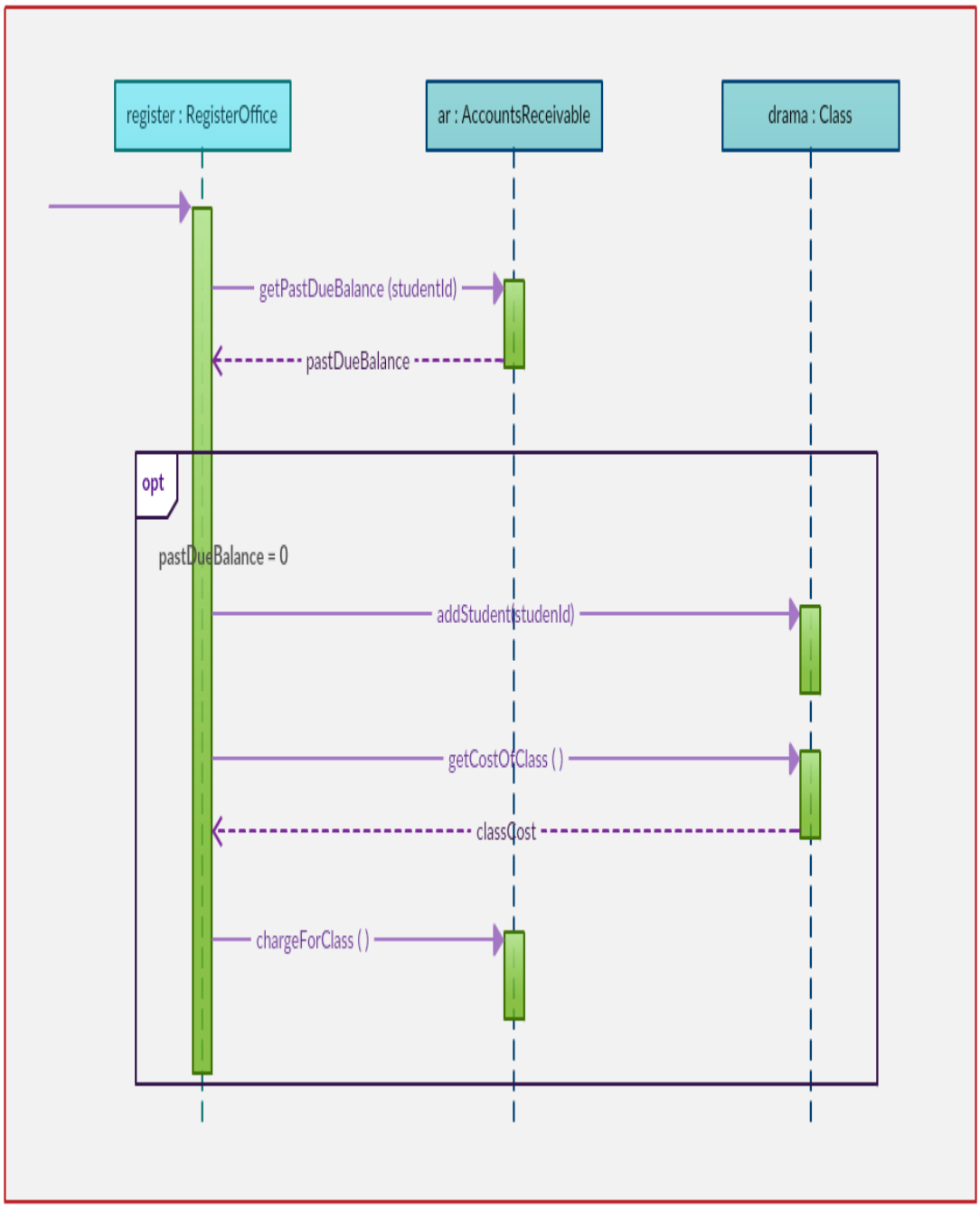
Click the image to edit it online

Online Examination System – Class Diagram (UML)

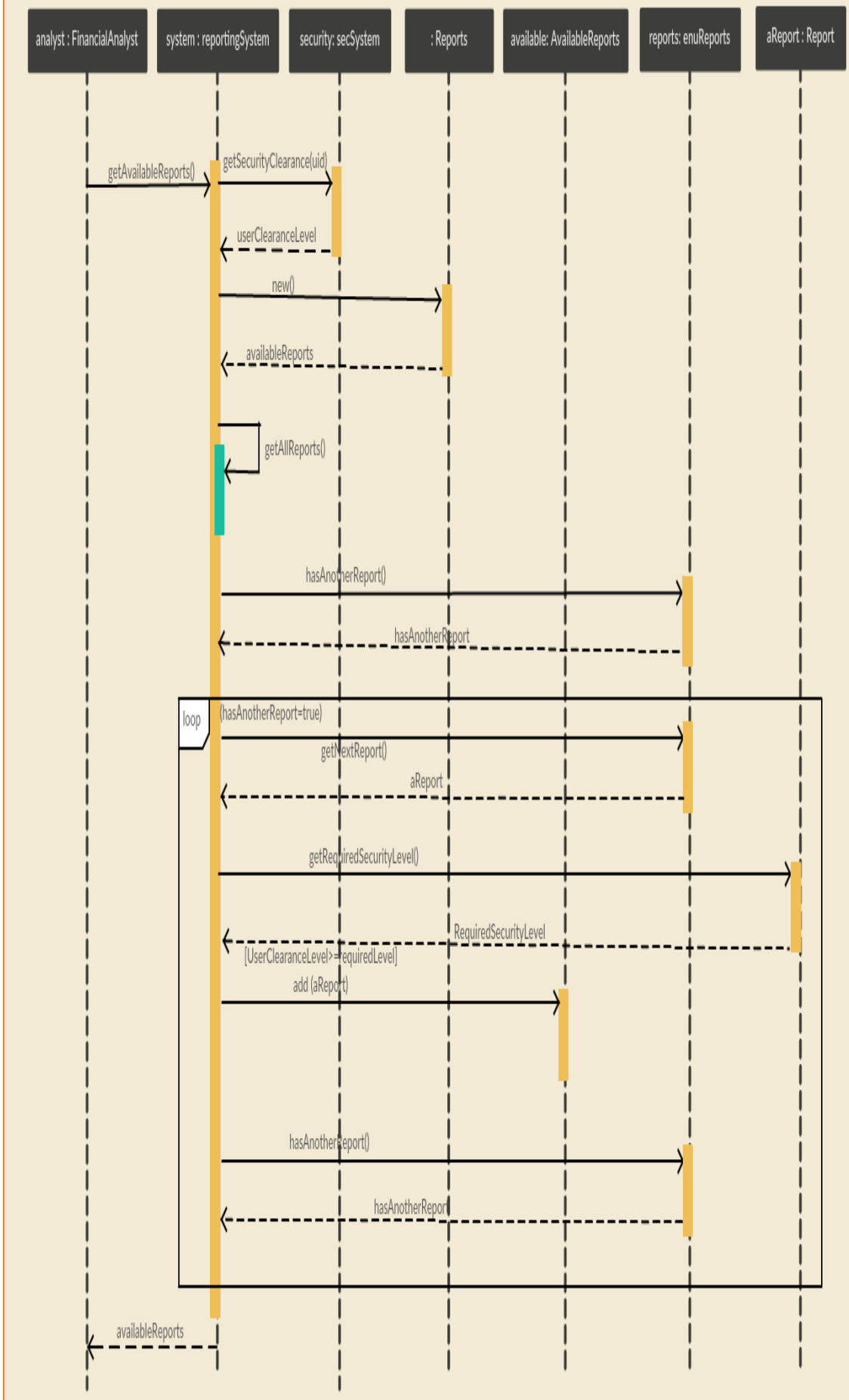
Sequence Diagram Example of a School Management System



Example of an Option Combination Fragment



Example of a Loop Sequence


analyst : FinancialAnalyst
system : reportingSystem
security: secSystem
: Reports
available: AvailableReports
reports: enuReports
aReport : Report
getAvailableReports()
getSecurityClearance(uid)
userClearanceLevel
new()
availableReports
getAllReports()
hasAnotherReport()
hasAnotherReport
loop
(hasAnotherReport=true)
getNextReport()
aReport
getRequiredSecurityLevel()
RequiredSecurityLevel
[UserClearanceLevel>=equiredLevel]
add (aReport)
hasAnotherReport()
hasAnotherReport
availableReports

Here are some more sequence diagram templates and examples that you can edit right away.

UML Collaboration Diagram

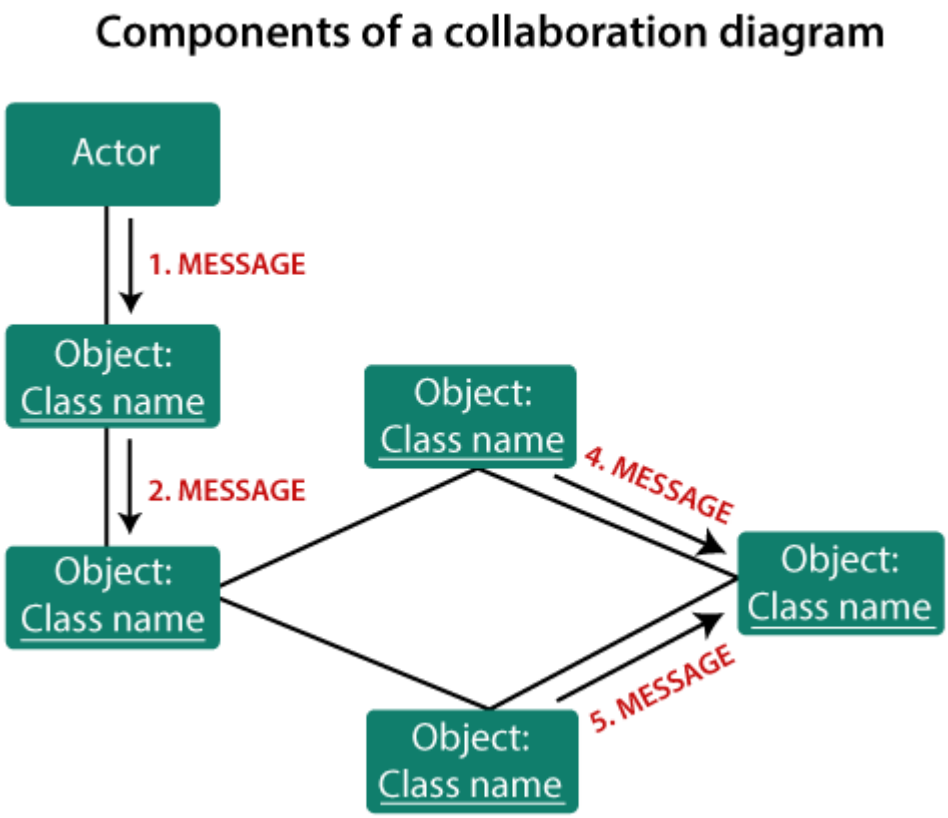
The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it 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.



When to use a Collaboration Diagram?

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). These diagrams can be used to portray the dynamic behavior of a particular use case and define the role of each object.

Collaboration diagrams are created by first identifying the structural elements required to carry out the functionality of an interaction. A model is then built using the relationships between those elements. Several vendors offer software for creating and editing collaboration diagrams.

When to use a collaboration diagram

Collaboration diagrams should be used when the relationships among objects are crucial to display. A few examples of instances where collaboration diagrams might be helpful include:

- Modeling collaborations, mechanisms or the structural organization within a system design.
- Providing an overview of collaborating objects within an object-oriented system.
- Exhibiting many alternative scenarios for the same use case.
- Demonstrating forward and reverse engineering.
- Capturing the passage of information between objects.
- Visualizing the complex logic behind an operation.

However, collaboration diagrams are best suited to the portrayal of simple interactions among relatively small numbers of objects. As the number of objects and messages grows, a collaboration diagram can become difficult to read and use efficiently. Additionally, collaboration diagrams typically exclude descriptive information, such as timing.

Collaboration vs sequence diagrams

In UML, the two types of interaction diagrams are collaboration and sequence diagrams. While both types use similar information, they display them in separate ways. Collaboration diagrams are used to visualize the structural organization of objects and their interactions. Sequence diagrams, on the other hand, focus on the order of messages that flow between objects. However, in most scenarios, a single figure is not sufficient in describing the behavior of a system and both figures are required.