Batch: A2	**Roll No.: 16010121045**
Experiment / assignment / tutorial No._______	

Title: To develop UML diagrams for a selected project

Aim: To learn and understand the way of creating various UML diagrams for requirement analysis

CO 2: Analyse the software requirements and Model the defined problem with the help of UML diagram.

Books/ Journals/ Websites referred:

1. Roger Pressman, “Software Engineering”, sixth edition, Tata McGraw Hill.
2. System Analysis & Design by Satzinger, Jackson and Burd, Cengage Learning, 2007
3. System Analysis and Design Methods by Jeffery l. Whitten, Lonnie D Bentley,McGraw Hill, 7th edition.
4. System Analysis and Design by Alan Dennis, Barbara H. Wixom, Roberta M. Roth,Wiley India 4th edition
5. http://en.wikipedia.org/wiki/Software_requirements_specification
6. http://en.wikipedia.org/wiki/Use_case

Pre Lab/ Prior Concepts:

In software and systems engineering, a **use case** is a list of steps, typically defining interactions between a role (known in Unified Modeling Language (UML) as an "actor") and a system, to achieve a goal. The actor can be a human or an external system.

In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. The detailed requirements may then be captured in Systems Modeling Language (SysML) or as contractual statements.

As an important requirement technique, use cases have been widely used in modern software engineering over the last two decades. Use case driven development is a key characteristic of process models and frameworks like Unified Process (UP), Rational Unified Process (RUP), Oracle Unified Method (OUM), etc. With its iterative and evolutionary nature, use case is also a good fit for agile development.

Use case diagrams

Use case diagrams belong to the category of behavioural diagram of UML diagrams. Use case diagrams aim to present a graphical overview of the functionality provided by the system. It consists of a set of actions (referred to as use cases) that the concerned system can perform, one or more actors, and dependencies among them.

Actor

An actor can be defined as “an object or set of objects, external to the system, which interacts with the system to get some meaningful work done. Actors could be human, devices, or even other systems.”

For example, consider the case where a customer withdraws cash from an ATM. Here, customer is a human actor.

Actors can be classified as below

Primary actor: They are principal users of the system, who fulfil their goal by availing some service from the system. For example, a customer uses an ATM to withdraw cash when he needs it. A customer is the primary actor here.

Supporting actor: They render some kind of service to the system. "Bank representatives", who replenishes the stock of cash, is such an example. It may be noted that replenishing stock of cash in an ATM is not the prime functionality of an ATM.

In a use case diagram primary actors are usually drawn on the top left side of the diagram.

Use Case

A use case is simply [1] a functionality provided by a system.

Continuing with the example of the ATM, withdraw cash is a functionality that the ATM provides. Therefore, this is a use case. Other possible use cases includes, check balance, change PIN, and so on.

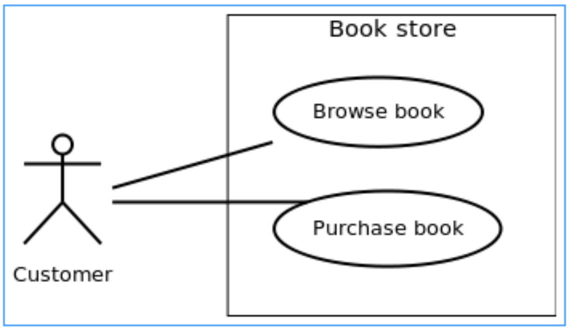
Use cases include both successful and unsuccessful scenarios of user interactions with the system. For example, authentication of a customer by the ATM would fail if he enters wrong PIN. In such case, an error message is displayed on the screen of the ATM.

Subject

Subject is simply [iii] the system under consideration. Use cases apply to a subject. For example, an ATM is a subject, having multiple use cases, and multiple actors interact with it. However, one should be careful of external systems interacting with the subject as actors.

Graphical Representation

An actor is represented by a stick figure and name of the actor is written below it. A use case is depicted by an ellipse and name of the use case is written inside it. The subject is shown by drawing a rectangle. Label for the system could be put inside it. Use cases are drawn inside the rectangle, and actors are drawn outside the rectangle.



Association between Actors and Use Cases
A use case is triggered by an actor. Actors and use cases are connected through binary associations indicating that the two communicates through message passing.

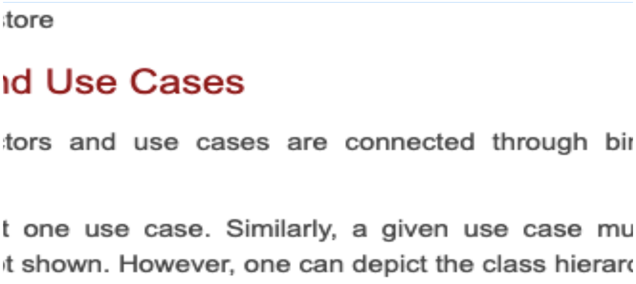
An actor must be associated with at least one use case. Similarly, a given use case must be associated with at least one actor. Association among the actors are usually not shown. However, one can depict the class hierarchy among actors.

Use Case Relationships
Three types of relationships exist among use cases:

Include relationship
Extend relationship
Use case generalization
Include Relationship
Include relationships are used to depict common behaviour that are shared by multiple use cases. This could be considered analogous to writing functions in a program in order to avoid repetition of writing the same code. Such a function would be called from different points within the program.

Example
For example, consider an email application. A user can send a new mail, reply to an email he has received, or forward an email. However, in each of these three cases, the

user must be logged in to perform those actions. Thus, we could have a login use case, which is included by compose mail, reply, and forward email use cases



tore

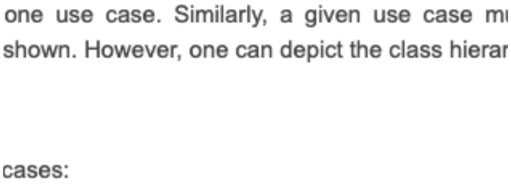
ıd Use Cases

tors and use cases are connected through bi

t one use case. Similarly, a given use case mu
t shown. However, one can depict the class hierar

Include relationship between use cases Notation
Include relationship is depicted by a dashed arrow with a «include» stereotype from the including use case to the included use case.

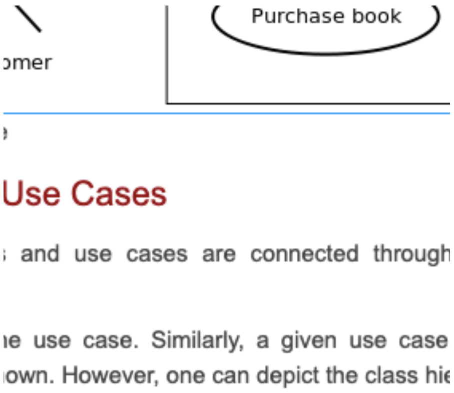
Extend Relationship
Use case extensions are used to depict any variation to an existing use case. They are used to the specify the changes required when any assumption made by the existing use case becomes false [iv, v].



one use case. Similarly, a given use case mı
shown. However, one can depict the class hierar

cases:

Generalization Relationship
Generalization relationship are used to represent the inheritance between use cases. A derived use case specializes some functionality it has already inherited from the base use case. Example to illustrate this, consider a graphical application that allows users to draw polygons. We could have a use case draw polygon. Now, rectangle is a particular instance of polygon having four sides at right angles to each other. So, the use case draw rectangle inherits the properties of the use case draw polygon and overrides it's drawing method. This is an example of generalization relationship. Similarly, a generalization relationship exists between draw rectangle and draw square use cases.



Use Cases

and use cases are connected through

e use case. Similarly, a given use case
own. However, one can depict the class hie

Generalization relationship is depicted by a solid arrow from the specialized (derived) use case to the more generalized (base) use case.

Identifying Actors
Given a problem statement, the actors could be identified by asking the following questions [2]:

Who gets most of the benefits from the system? (The answer would lead to the identification of the primary actor)
Who keeps the system working? (This will help to identify a list of potential users)
What other software / hardware does the system interact with?
Any interface (interaction) between the concerned system and any other system?
Identifying Use cases
Once the primary and secondary actors have been identified, we have to find out their goals i.e. what are the functionality they can obtain from the system. Any use case name should start with a verb like, "Check balance".

Guidelines for drawing Use Case diagrams
Following general guidelines could be kept in mind while trying to draw a use case diagram :

- Determine the system boundary
- Ensure that individual actors have well-defined purpose
- Use cases identified should let some meaningful work done by the actors
- Associate the actors and use cases -- there shouldn't be any actor or use case floating without any connection
- Use include relationship to encapsulate common behaviour among use cases , if any

Capturing the dynamic view of a system is very important for a developer to develop the logic for a system. State chart diagrams and activity diagrams are two popular UML diagram to visualize the dynamic behaviour of an information system.

Different components of activity diagram and state chart diagram can be used to represent the dynamic nature of an information system.

Activity Diagrams

Activity diagrams fall under the category of behavioural diagrams in Unified Modelling Language. It is a high level diagram used to visually represent the flow of control in a system. It has similarities with traditional flow charts. However, it is more powerful than a simple flow chart since it can represent various other concepts like concurrent activities, their joining, and so on [vii, viii].

Activity diagrams, however, cannot depict the message passing among related objects. As such, it can't be directly translated into code. These kind of diagrams are suitable for confirming the logic to be implemented with the business users. These diagrams are typically used when the business logic is complex. In simple scenarios it can be avoided entirely [ix].

Components of an Activity Diagram
Below we describe the building blocks of an activity diagram.

Activity
An activity denotes a particular action taken in the logical flow of control. This could simply be invocation of a mathematical function, alter an object's properties and so on [x]. An activity is represented with a rounded rectangle, as shown in table-01. A label inside the rectangle identifies the corresponding activity.

There are two special type of activity nodes: initial and final. They are represented with a filled circle, and a filled in circle with a border respectively (table-01). Initial node represents the starting point of a flow in an activity diagram. There could be multiple initial nodes, which means that invoking that particular activity diagram would initiate multiple flows.

A final node represents the end point of all activities. Like an initial node, there could be multiple final nodes. Any transition reaching a final node would stop all activities.

Flow
A flow (also termed as edge, or transition) is represented with a directed arrow. This is used to depict transfer of control from one activity to another, or to other types of components, as we will see below. A flow is often accompanied with a label, called the guard condition, indicating the necessary condition for the transition to happen. The syntax to depict it is [guard condition].

Decision
A decision node, represented with a diamond, is a point where a single flow enters and two or more flows leave. The control flow can follow only one of the outgoing paths. The outgoing edges often have guard conditions indicating true-false or if-then-else

conditions. However, they can be omitted in obvious cases. The input edge could also have guard conditions. Alternately, a note can be attached to the decision node indicating the condition to be tested.

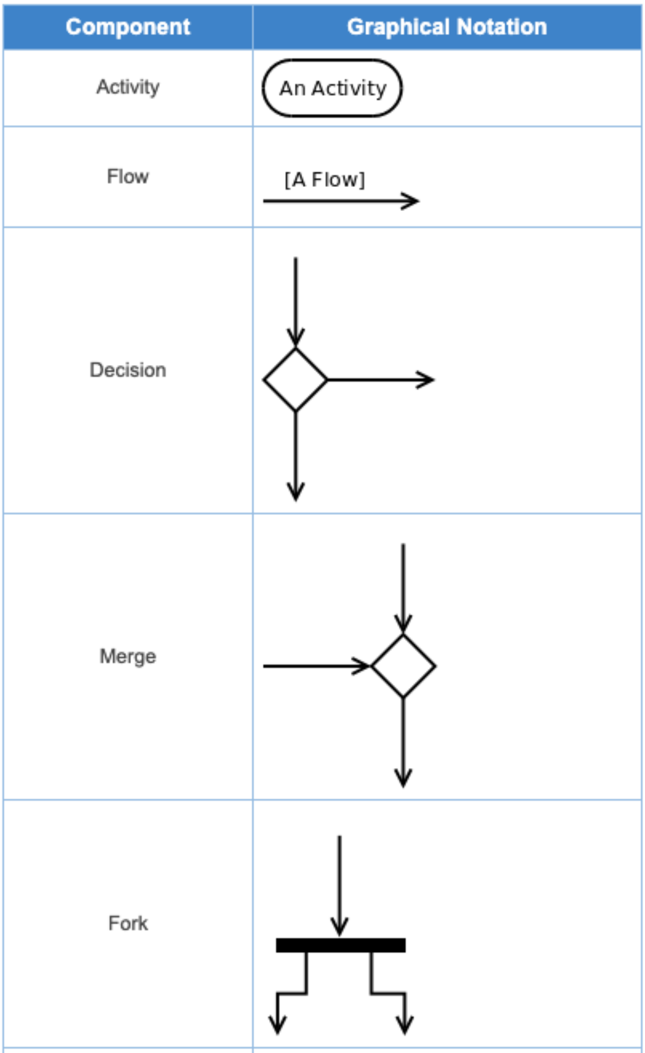
Merge
This is represented with a diamond shape, with two or more flows entering, and a single flow leaving out. A merge node represents the point where at least a single control should reach before further processing could continue.

Fork
Fork is a point where parallel activities begin. For example, when a student has been registered with a college, he can in parallel apply for student ID card and library card. A fork is graphically depicted with a black bar, with a single flow entering and multiple flows leaving out.

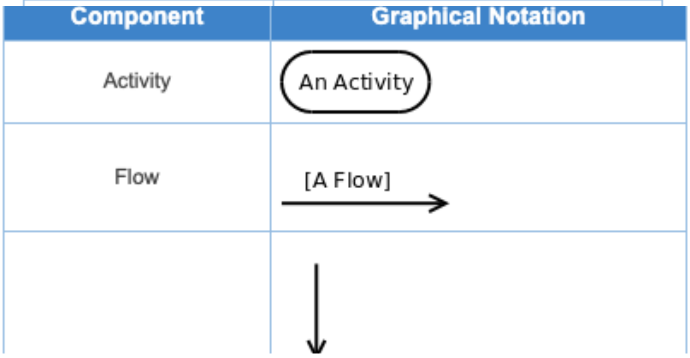
Join
A join is depicted with a black bar, with multiple input flows, but a single output flow. Physically it represents the synchronization of all concurrent activities. Unlike a merge, in case of a join all of the incoming controls must be completed before any further progress could be made. For example, a sales order is closed only when the customer has receive the product, and the sales company has received it's payment.

Note
UML allows attaching a note to different components of a diagram to present some textual information. The information could simply be a comment or may be some constraint. A note can be attached to a decision point, for example, to indicate the branching criteria.

Partition
Different components of an activity diagram can be logically grouped into different areas, called partitions or swim-lanes. They often correspond to different units of an organization or different actors. The drawing area can be partitioned into multiple compartments using vertical (or horizontal) parallel lines. Partitions in an activity diagram are not mandatory.



Component	Graphical Notation
Activity	An Activity
Flow	[A Flow]
Decision	
Merge	
Fork	



Component	Graphical Notation
Activity	An Activity
Flow	[A Flow]

The following general guidelines could be followed to pictorially represent a complex logic.

- Identify tiny pieces of work being performed by the system
- Identify the next logical activity that should be performed
- Think about all those conditions that should be made, and all those constraints that should be satisfied, before one can move to the next activity
- Put non-trivial guard conditions on the edges to avoid confusion

In UML, class diagrams are one of six types of structural diagram. Class diagrams are fundamental to the object modelling process and model the static structure of a system. Depending on the complexity of a system, you can use a single class diagram to model an entire system, or you can use several class diagrams to model the components of a system.

Class diagrams are the blueprints of your system or subsystem. You can use class diagrams to model the objects that make up the system, to display the relationships between the objects, and to describe what those objects do and the services that they provide.

In its basic form, an activity diagram is a simple and intuitive illustration of what happens in a workflow, what activities can be done in parallel, and whether there are alternative paths through the workflow. Activity diagrams as defined in the Unified Modeling Language are derived from various techniques to visually illustrate workflows. Activity diagrams are used to visualize the workflow of a business use case. A complete workflow description will have a basic flow, and one or several alternative flows. This workflow has a structure that we can define textually, using informal if, if-then-else, or does-until statements of various kinds. For a simple workflow with a simple structure such textual definitions may be quite sufficient, but in the case of more complex structures, activity diagrams help to clarify and make more apparent what the workflow is. Historically, activity diagramming techniques have mostly been used in the business process modeling domain, but this article will also briefly discuss how you can use it in the system modeling domain.

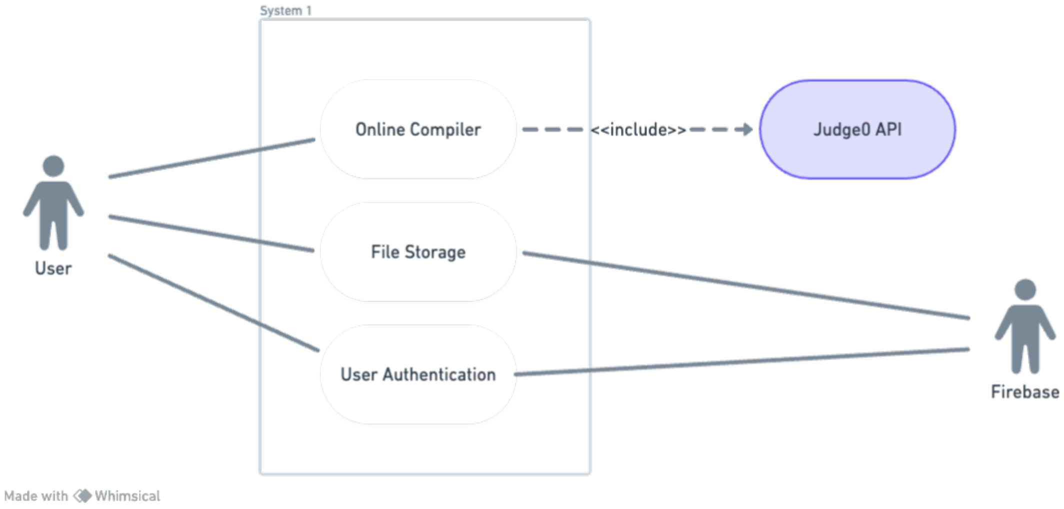
A sequence diagram is a graphical view of a scenario that shows object interaction in a time-based sequence of what happens first, what happens next.
Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces.

Requirement Modeling:
Significance of every diagram is to be written

1. **Use Case**

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. UML use case diagrams are ideal for:

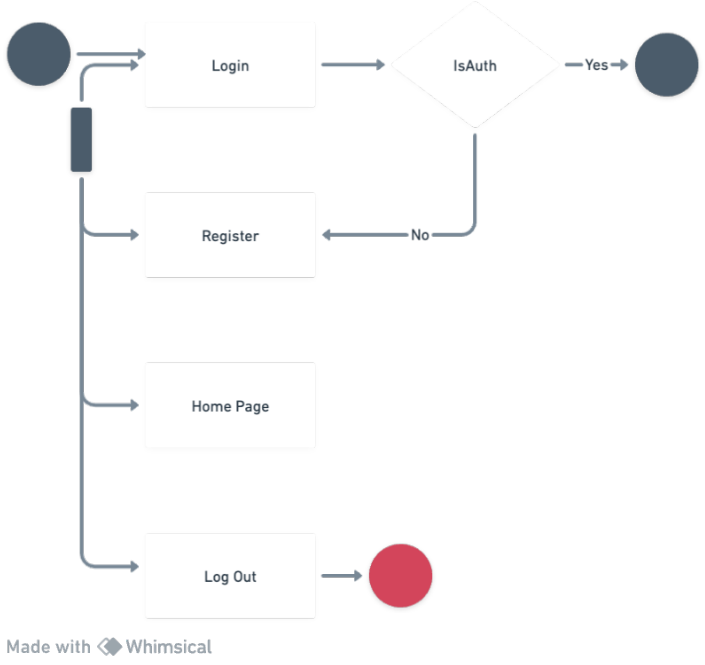
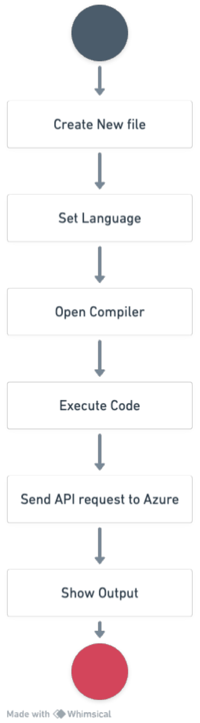
- Representing the goals of system-user interactions
- Defining and organizing functional requirements in a system
- Specifying the context and requirements of a system
- Modeling the basic flow of events in a use case



2. Activity Diagram

Activity diagrams help people on the business and development sides of an organization come together to understand the same process and behavior. You'll use a set of specialized symbols—including those used for starting, ending, merging, or receiving steps in the flow—to make an activity diagram. Activity diagrams present a number of benefits to users. It can be used to:

- Demonstrate the logic of an algorithm.
- Describe the steps performed in a UML use case.
- Illustrate a business process or workflow between users and the system.
- Simplify and improve any process by clarifying complicated use cases.
- Model software architecture elements, such as method, function, and operation.

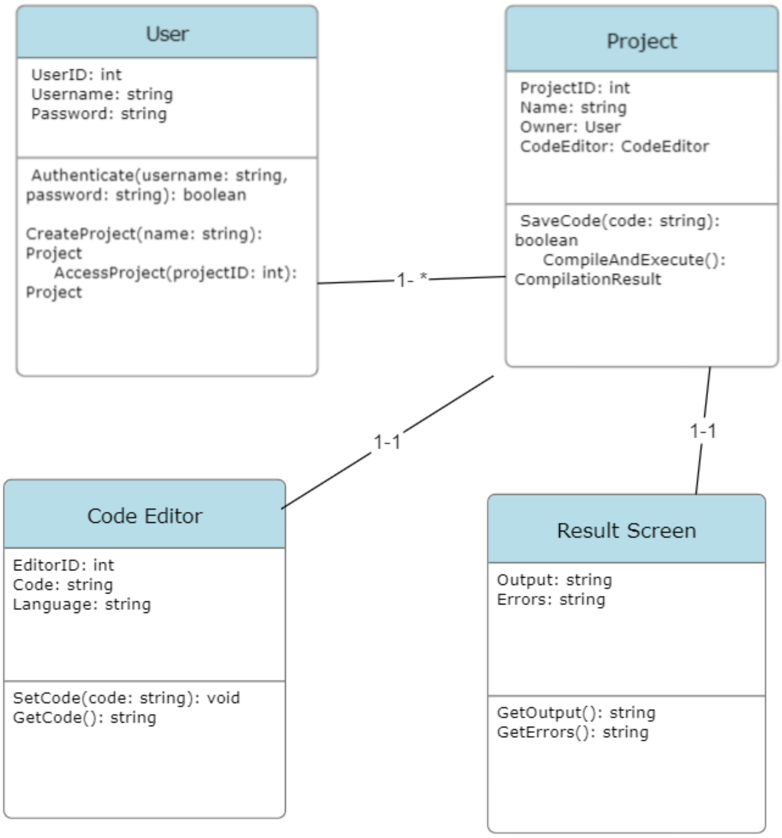
Home	Logic
 Login IsAuth Yes No Register Home Page Log Out Made with Whimsical	 Create New file Set Language Open Compiler Execute Code Send API request to Azure Show Output Made with Whimsical

3. Class Diagram

Class diagrams are one of the most useful types of diagrams in UML as they clearly map out the structure of a particular system by modeling its classes, attributes, operations, and relationships between objects.
Since classes are the building block of objects, class diagrams are the building blocks of UML. The various components in a class diagram can represent the classes that will be programmed, the main objects, or the interactions between classes and objects.

Use UML class diagrams to:

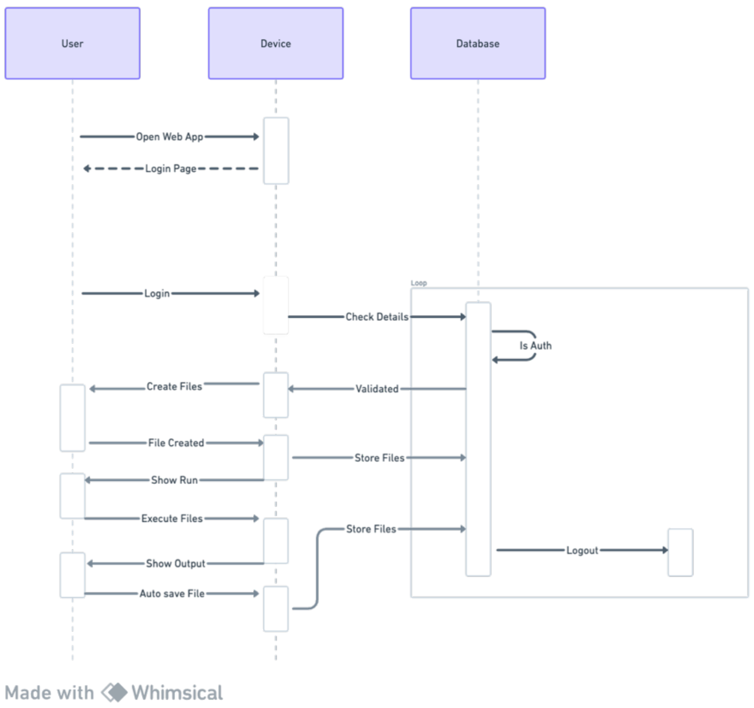
- Illustrate data models for information systems, no matter how simple or complex.
- Better understand the general overview of the schematics of an application.
- Visually express any specific needs of a system and disseminate that information throughout the business.
- Create detailed charts that highlight any specific code needed to be programmed and implemented to the described structure.
- Provide an implementation-independent description of types used in a system that are later passed between its components.



4. Sequence

A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios. Benefits of sequence diagrams

- Sequence diagrams can be useful references for businesses and other organizations. Try drawing a sequence diagram to:
- Represent the details of a UML use case.
- Model the logic of a sophisticated procedure, function, or operation.
- See how objects and components interact with each other to complete a process.
- Plan and understand the detailed functionality of an existing or future scenario. The following scenarios are ideal for using a sequence diagram:
- Usage scenario: A usage scenario is a diagram of how your system could potentially be used. It's a great way to make sure that you have worked through the logic of every usage scenario for the system.
- Method logic: Just as you might use a UML sequence diagram to explore the logic of a use case, you can use it to explore the logic of any function, procedure, or complex process.
- Service logic: If you consider a service to be a high-level method used by different clients, a sequence diagram is an ideal way to map that out.



Conclusion: Through this experiment we learnt the different modelling diagrams like use case, activity, class and sequence diagram. We implemented this diagram for our chosen mini project and learnt their features, pros and cons.

Post Lab Descriptive Questions:

1. Where do use cases fit in the software development life cycle?

Use cases are a critical part of the software development life cycle, particularly in the early stages when requirements are being gathered and analyzed. Here's where they fit in:

- **Requirements Gathering:** Use cases are typically created during the requirements gathering phase. They help capture and define the functional requirements of the system from a user's perspective. Use cases are often written in a natural language, making them accessible to non-technical stakeholders.

- **System Design:** Use cases can influence the system's high-level design. They help in identifying the major components and interactions within the system. Use cases can also aid in defining the system's architecture and deciding on the technologies to be used.

- **Development:** During the development phase, use cases serve as the basis for creating detailed functional specifications, user stories, or test cases. Developers use these documents to build the software system according to the specified functionality.

- **Testing:** Use cases are essential for creating test cases. Testers verify that the software behaves according to the use cases, ensuring that it meets the intended requirements.

- **Documentation:** Use cases provide valuable documentation for the system, serving as a reference for how the system is expected to work. They are also helpful for training end-users.

- **Maintenance:** Use cases continue to be relevant in the maintenance phase, helping developers and testers understand the expected behavior of the system, making it easier to identify and fix issues.

2. Compare sequence diagram with collaboration diagram. Explain pros and cons of each.

Sequence Diagram:

- **Pros:**
  - **Time Sequencing:** Sequence diagrams show the order and timing of interactions between objects, making them ideal for understanding the chronological flow of messages in a system.
  - **Clarity:** They are often easier to read and understand than collaboration diagrams, especially for complex interactions involving many objects.
  - **Focus on Object Behavior:** Sequence diagrams highlight how objects collaborate to achieve a specific task, emphasizing the behavioral aspects of the system.
- **Cons:**
  - **Limited in Representing Structural Information:** Sequence diagrams are not designed to show the structural relationships between objects as clearly as collaboration diagrams.
  - **Can Become Complex:** In large systems, sequence diagrams can become intricate and difficult to manage, potentially leading to confusion.

Collaboration Diagram (Communication Diagram):

- **Pros:**
  - **Clear Structural Relationships:** Collaboration diagrams excel at representing the structural relationships between objects, helping stakeholders understand the object composition of a system.

    - **Simplified Communication Paths:** They simplify the representation of communication paths by removing the emphasis on strict time sequencing.
    - **Readability for Structural Information:** Collaboration diagrams are often easier to read when you want to focus on object relationships.
- **Cons:**
    - **Less Emphasis on Timing:** Collaboration diagrams do not convey the precise timing of messages as clearly as sequence diagrams.
    - **Complex Interactions:** They can become cluttered and less intuitive when representing complex sequences of interactions.

3. List different notations used in Class diagram with example

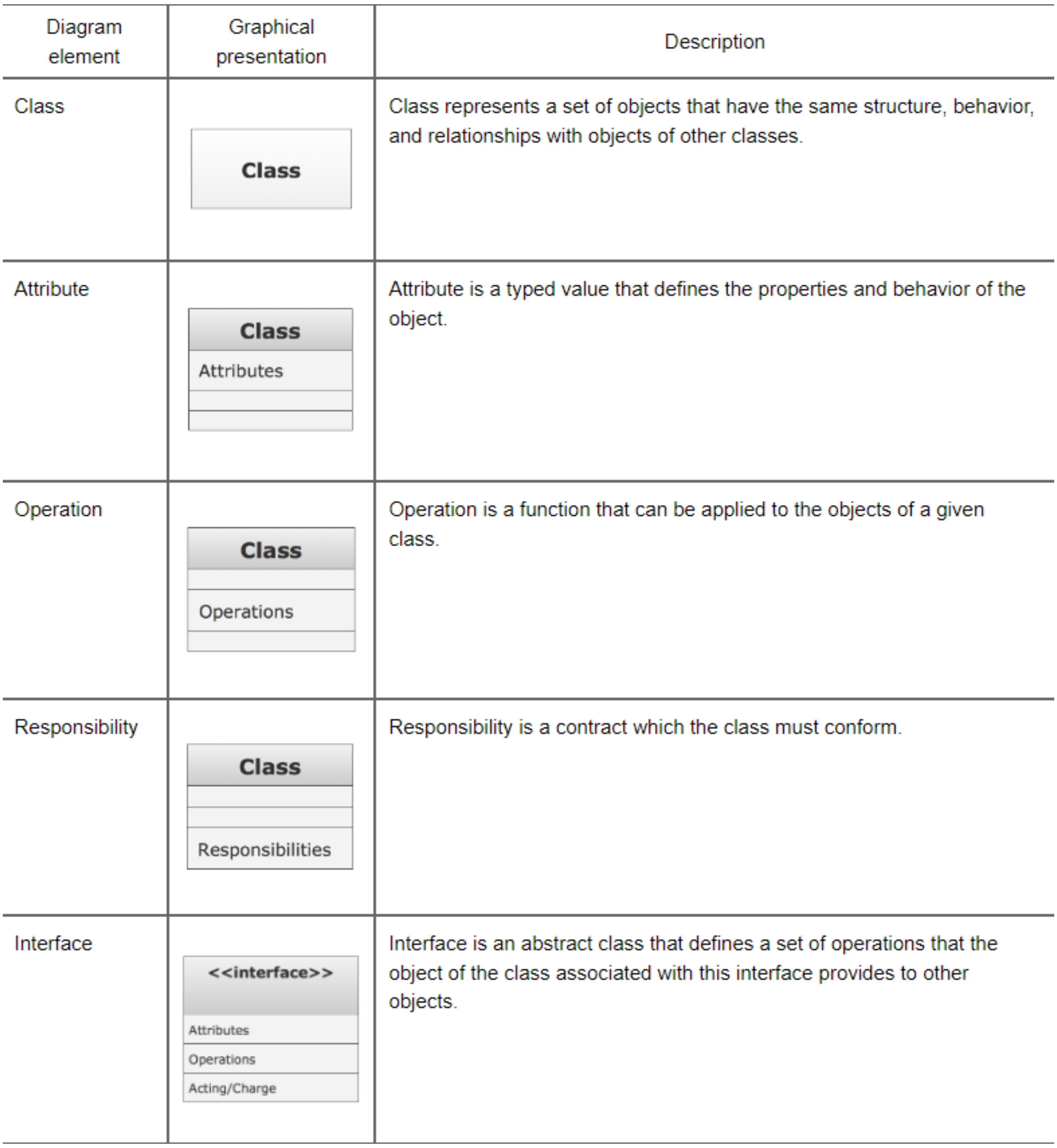
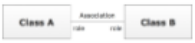
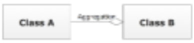
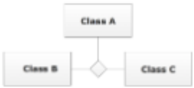
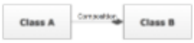
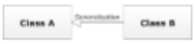
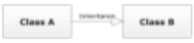
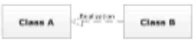
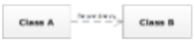
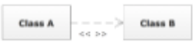
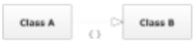


Diagram element	Graphical presentation	Description
Class	Class	Class represents a set of objects that have the same structure, behavior, and relationships with objects of other classes.
Attribute	Class Attributes	Attribute is a typed value that defines the properties and behavior of the object.
Operation	Class Operations	Operation is a function that can be applied to the objects of a given class.
Responsibility	Class Responsibilities	Responsibility is a contract which the class must conform.
Interface	<<interface>> Attributes Operations Acting/Charge	Interface is an abstract class that defines a set of operations that the object of the class associated with this interface provides to other objects.

Association		Association is a relationship that connect two classes.
Aggregation		Aggregation is an association with the relation between the whole and its parts, the relation when one class is a certain entity that includes the other entities as components.
N-ary Association		N-ary association represents two or more aggregations.
Composition		Composition is a strong variant of aggregation when parts cannot be separately of the whole.
Generalization		Generalization ia an association between the more general classifier and the more special classifier.
Inheritance		Inheritance is a relationship when a child object or class assumes all properties of his parent object or class.
Realization		Realization is a relationship between interfaces and classes or components that realize them.
Dependency		Dependency is a relationship when some changes of one element of the model can need the change of another dependent element.
<< >>		Allows to define the properties of the dependency relationship between classes or classes and packages.
{ }		Allows to indicate the additional properties of association.