Batch: A2 **Roll No.:** 16010122041

Experiment No. 5

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: To develop UML diagrams for selected project

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

CO: 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:

Use Case Diagram:

- A use case diagram shows the interaction between users (actors) and the system. It highlights the different functionalities (use cases) the system provides and who interacts with them.
- In your project, the use case diagram helps identify the key features of the system and the primary actors involved. It provides a high-level view of what the system will do, allowing stakeholders to understand the system's capabilities and how users will interact with it.

Activity Diagram:

- An activity diagram represents the workflow or the sequence of activities in a system. It illustrates the dynamic aspects of the system, showing how different activities are connected and the flow of control from one activity to another.
- For your project, the activity diagram helps map out the processes or workflows within the system. It allows you to visualize how the system behaves during different operations, such as how a user might navigate through the system or how different components interact during a process.

Sequence Diagram:

- A sequence diagram focuses on the time-ordering of messages or interactions between objects or components in the system. It shows how objects interact with each other over time to carry out a particular functionality.
- In your project, the sequence diagram is used to detail the interactions between objects for specific use cases. It helps in understanding the flow of messages between objects, the order of operations, and how the system's components work together to achieve a particular outcome.

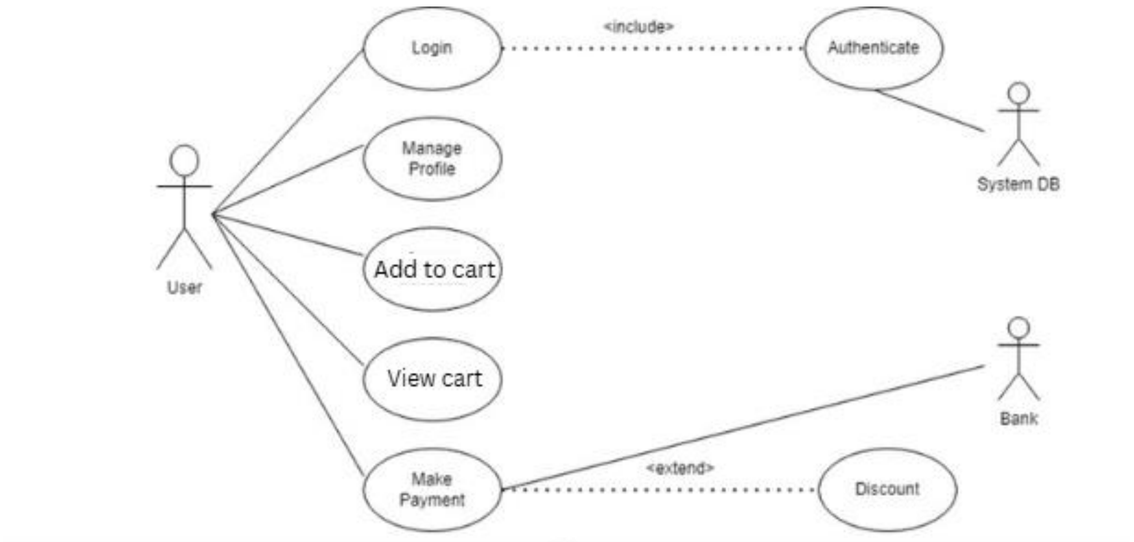
Class Diagram:

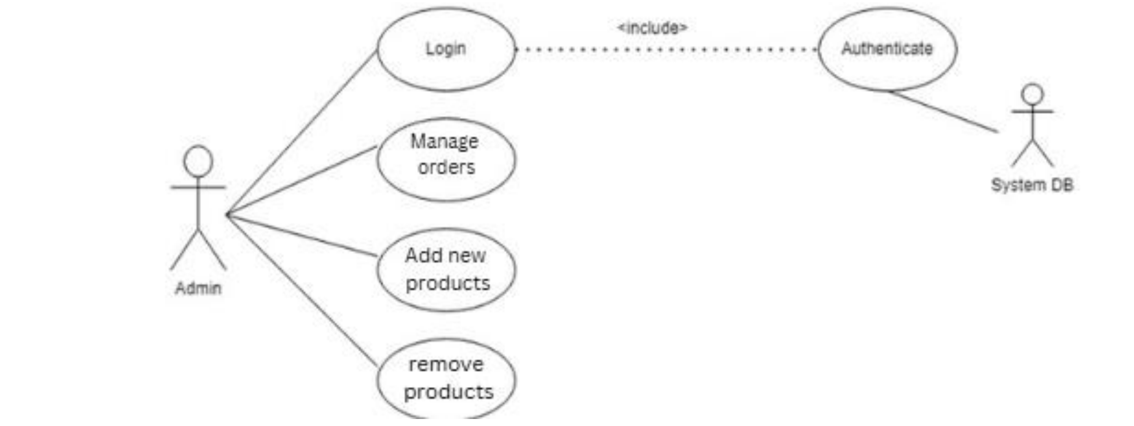
- A class diagram describes the static structure of the system by showing the system's classes, their attributes, operations, and the relationships between them.
- For your project, the class diagram serves as the blueprint for the system's code. It defines the system's data structure and the relationships between different entities (classes). This diagram is essential for understanding how the system's data is organized and how different classes interact to perform the system's functionalities.

Developing these UML diagrams for your project provides a comprehensive view of both the static and dynamic aspects of the system. They are crucial for the design phase, ensuring that all stakeholders have a clear understanding of the system's architecture and behavior. By creating these diagrams, you lay the foundation for a well-structured, maintainable, and scalable software system.

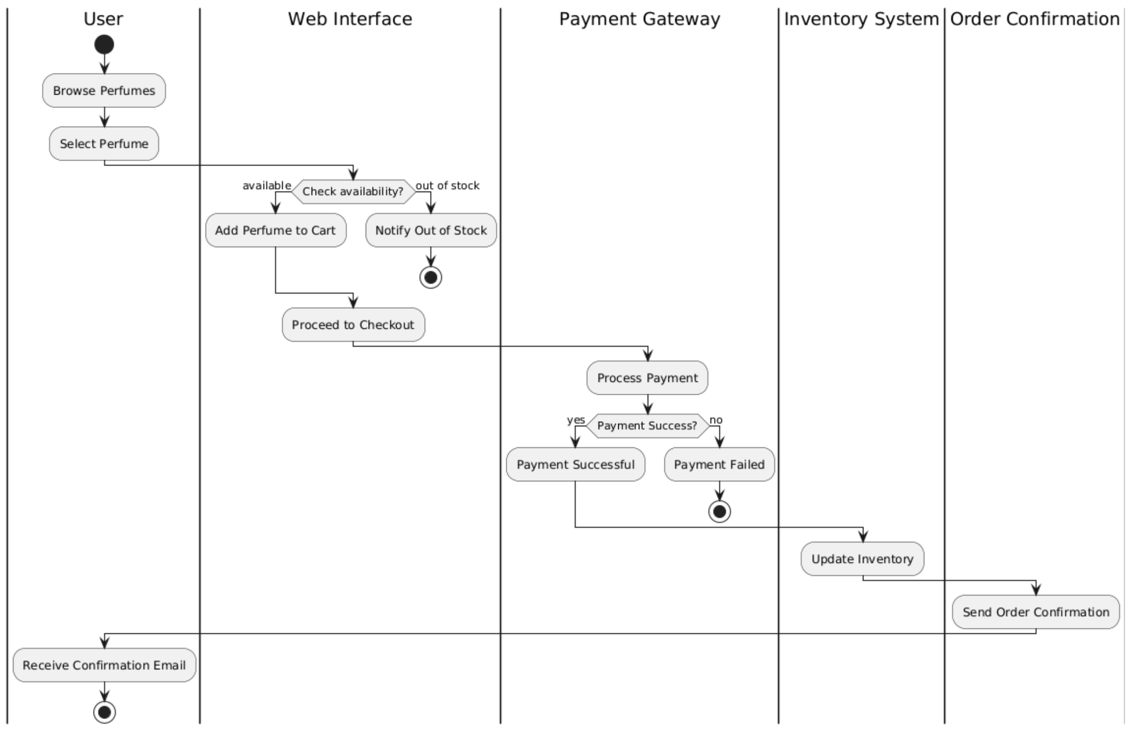
Diagram for your system:

Use case:

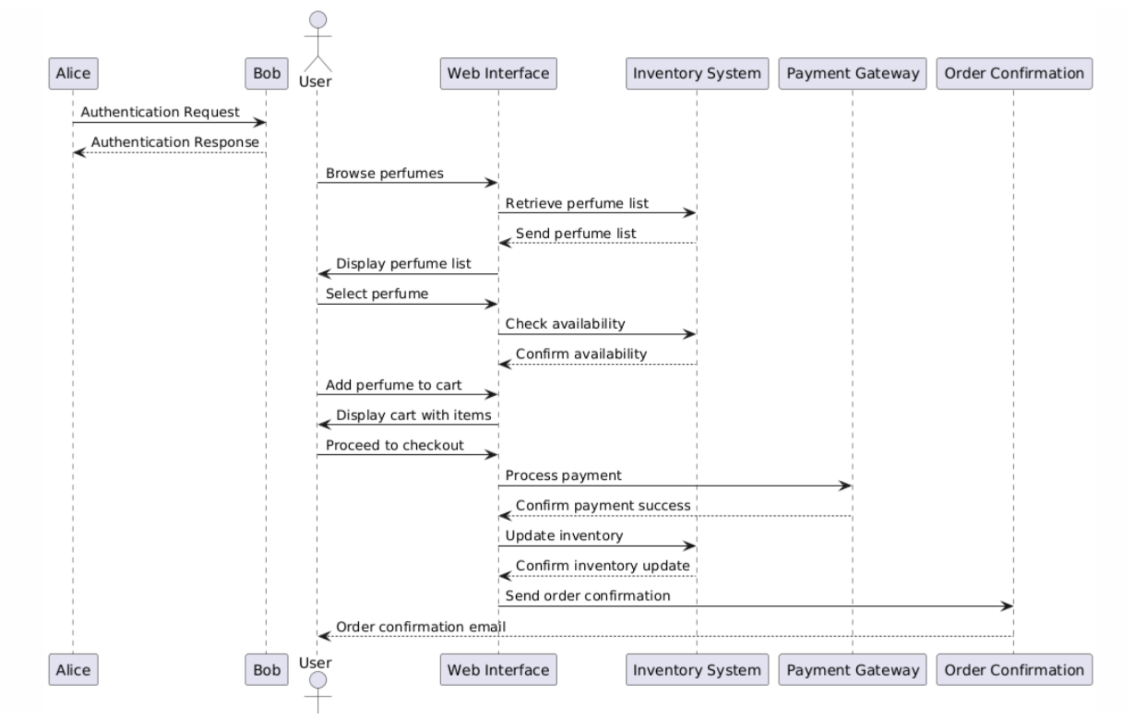




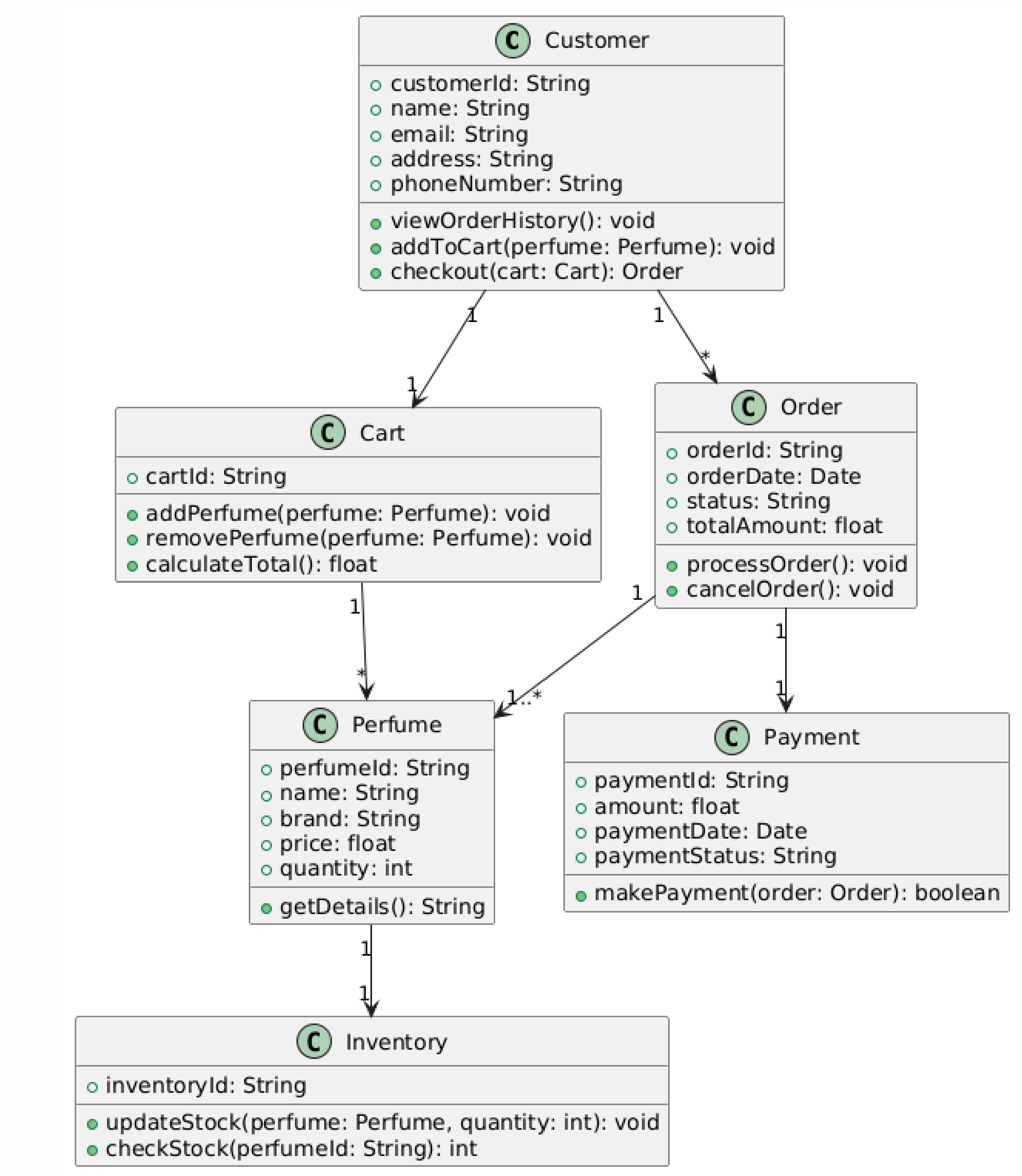
Activity diagram:



Sequence diagram:



Class diagram:



Conclusion:

Post Lab Descriptive Questions:

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

Use cases play a crucial role in the software development life cycle (SDLC) primarilyduring the requirements gathering and analysis phase. Here's how they fit in:

- **Requirements Gathering**: Use cases help capture functional requirements by describing how users will interact with the system. They provide a clear and structured way to document user needs.
- **Analysis and Design**: Use cases aid in understanding user interactions, whichcan influence system design. They help identify system components and workflows, making it easier for developers and designers to visualize how thesystem will function.
- **Validation**: During testing, use cases serve as a reference to ensure that the system meets user requirements. Test cases can be derived from use cases tovalidate functionality.
- **Communication**: Use cases provide a common language for stakeholders, including developers, designers, testers, and clients, helping to align everyone'sunderstanding of system functionality.
- **Maintenance**: They can also be useful in later stages of the SDLC, such as maintenance, to understand how changes might affect existing functionalities.

Overall, use cases are integral to ensuring that the software meets user needsthroughout the development process.

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

Sequence Diagram

Description: Sequence diagrams illustrate how objects interact in a specific sequenceover time. They show the order of messages exchanged between objects, along with lifelines that represent the objects involved.

Pros:

- **Time-Ordered**: Clearly shows the timing and order of messages, making it easier to understand the sequence of events.
- **Focus on Behavior**: Excellent for modeling complex interactions and understanding the dynamics of the system.
- **Lifelines**: Lifelines help visualize object existence and message flow over time.

Cons:

- **Complexity in Large Systems**: Can become cluttered and hard to read with manyobjects or messages, making it challenging to follow.
- **Less Emphasis on Structure**: Doesn't inherently show the relationships or roles of objects as clearly as collaboration diagrams.

Collaboration Diagram

Description: Collaboration diagrams (also known as communication diagrams) emphasize the relationships between objects and how they collaborate to fulfill a

particular function. They depict messages exchanged but focus on the structure ratherthan the time sequence.

Pros:

- **Structural View**: Clearly shows how objects are interconnected and their roles within the system, which can aid in understanding the overall architecture.
- **Simplicity**: Generally easier to read and understand for smaller interactions or simpler systems.
- **Flexible Layout**: Can be arranged in various ways to highlight different aspects of collaboration.

Cons:

- **Less Clarity on Timing**: Does not provide a clear view of the timing or order of message exchanges, which can make understanding the flow of interactions difficult.
- **Complexity with Many Interactions**: Can become complex and hard to follow whendepicting many objects and interactions, similar to sequence diagrams.

Conclusion

In summary, **sequence diagrams** are ideal for visualizing time-based interactions,while **collaboration diagrams** provide a clearer view of object relationships. The

choice between them depends on the specific needs of the project—whether the focus is on the timing of messages or the structural collaboration among objects. Often, both diagrams can complement each other to give a fuller picture of the system’s behavior.

3. List different notations used in Class diagram with example

1. Class Notation

- **Description**: A rectangle divided into three sections:
  - **Top**: Class name
  - **Middle**: Attributes (fields)
  - **Bottom**: Methods (operations)
- **Example**: A class called "Person" with attributes like "name" and "age" and methods like "getName()" and "getAge()".

2. Attributes Notation

- **Description**: Attributes include visibility indicators:
  - + for public
  - – for private
  - # for protected
- **Example**: A private attribute might be "address: String".

3. Methods Notation

- **Description**: Similar to attributes, methods also have visibility indicators.
- **Example**: A public method could be "setName(name: String): void".

4. Association Notation

- **Description**: A solid line connecting two classes, representing a relationship. Multiplicity is shown at both ends (e.g., 1, 0..1, *).
- **Example**: A "Person" class associated with multiple "Address" instances.

5. Inheritance Notation

- **Description**: A solid line with a closed arrowhead pointing to the parent class.
- **Example**: A "Dog" class that inherits from an "Animal" class.

6. Interface Notation

- **Description**: Represented by a rectangle with the «interface» stereotype or sometimes by a circle.
- **Example**: An interface called "IAnimal" with a method "makeSound()".

7. Dependency Notation

- **Description**: A dashed line with an open arrowhead, indicating one class depends on another.
- **Example**: A "Dog" class depending on the "IAnimal" interface.

8. Aggregation Notation

- **Description**: A line with a hollow diamond at the aggregate end, indicating a whole-part relationship.
- **Example**: A "School" that aggregates multiple "Student" instances.

9. Composition Notation

- **Description**: A line with a filled diamond at the composite end, indicating a stronger whole-part relationship.
- **Example**: A "House" that consists of multiple "Room" instances.

4. Modeling UML Use Case, Class diagram, Sequence Diagram.

   a) Virtual Lab Link: http://vlabs.iitkgp.ernet.in/se/3/exercise/
   b) Virtual Lab Link: http://vlabs.iitkgp.ernet.in/se/5/exercise/
   c) Virtual Lab Link: http://vlabs.iitkgp.ernet.in/se/6/
   d) **Virtual Lab Link: http://vlabs.iitkgp.ernet.in/se/7/**