|  |  |  |
| --- | --- | --- |
| **Academic Year:** 2024-25 | **Year:** Second Year | **Term:** II |
| **PRN No.: 1012412079** | **Name: Ratnajeet Patil** | |
| **Subject:** Software Engineering | | |
| **Assignment No.**: 8 |  | |
| **Date:** |  | |

**Lab Assignment: 08**

**Title**: To identify classes and create class diagram for a given problem.

**1. What is a Class Diagram?**

A **Class Diagram** is a type of static structure diagram in UML (Unified Modeling Language) that shows the structure of a system by depicting its classes, their attributes, methods (or operations), and the relationships between the classes. It is used to model the static view of an application.

Class diagrams are typically used in object-oriented design and can describe the classes, their instances, and the relationships between them in a system.

**2. Structure of a Class Diagram**

Each class in a class diagram has the following structure:

* **Name**: The name of the class (e.g., Customer, PaymentProcessor).
* **Attributes**: The properties or data members of the class (e.g., accountNumber: String, balance: double).
* **Methods**: The functions or operations that the class can perform (e.g., deposit(), withdraw()).

The class diagram typically uses the following visibility for attributes and methods:

* **Public**: Denoted by a + sign, meaning the attribute or method is accessible from any other class.
* **Private**: Denoted by a - sign, meaning the attribute or method is only accessible within the class itself.
* **Protected**: Denoted by a # sign, meaning the attribute or method is accessible within the class and its subclasses.

**3. Key Concepts in Class Diagrams:**

**Public, Private, and Protected Attributes:**

* **Public** attributes and methods are accessible from outside the class (e.g., +accountNumber: String).
* **Private** attributes and methods are only accessible within the class (e.g., -balance: double).
* **Protected** attributes and methods are accessible within the class and any subclass (e.g., #interestRate: float).

**Generalization:**

* Generalization is represented by an arrow with a hollow triangle at the base pointing to the parent class. It shows an "is-a" relationship between a subclass and its superclass (e.g., Manager is a subclass of Employee).

**Association:**

* Association represents a relationship between two classes. It is shown with a solid line. It can have multiplicity (e.g., one-to-many, many-to-many) and may include labels to describe the relationship (e.g., customer "has" account).

**Composition:**

* Composition is a form of association that represents a strong "whole-part" relationship, where the part cannot exist without the whole. It is shown by a filled diamond at the composite (whole) end.

**Aggregation:**

* Aggregation is a "whole-part" relationship where the part can exist independently of the whole. It is represented by an empty diamond at the aggregate (whole) end.

**Multiplicity:**

* Multiplicity indicates how many instances of one class can be associated with instances of another class. It is represented by numbers (e.g., 1, 0..1, 1..\*).

