The University of Azad Jammu & Kashmir, Muzaffarabad

Object Oriented Programming

UML Class Diagram

Student Name:	Shahzad Ahmed Awan
Roll No:	2024-SE-15
Course Title:	Object Oriented Programming
Course Code:	CS-1204
Instructor:	Engr. Awais Rathore
Submission Date:	19-May-2025

Department of Software Engineering

Application Used in Assignment

StarUML

Assignment: Introduction to UML Class Diagram

This assignment aims to enhance our understanding of UML class diagrams, which are fundamental in software design. By learning to create these diagrams, we will develop the ability to structure and represent system components effectively.

Table 1: Key UML Class Diagram Symbols

Element	Symbol / Format	Purpose / Description
Class	Rectangle with 3 sections	Name, attributes, and methods.
Attribute	`visibility name: type`	Defines class properties (e.g., `- name: string`).
Method	`visibility name(params): returnType`	Defines behavior (e.g., `+ deposit(amount: double): void`).
Public	`+`	Accessible from anywhere.
Private	`-`	Accessible only within the class.
Protected	`#`	Accessible in the class and its subclasses.
Static Member	*Underlined* name	Belongs to the class rather than an instance.
Generalization	Solid line with hollow triangle (–▷)	Represents inheritance ("is-a").
Association	Solid line	Indicates a relationship or usage ("has-a").
Multiplicity	`1`, `0..1`, `*`, `1..*`	Indicates how many objects participate in the relationship.

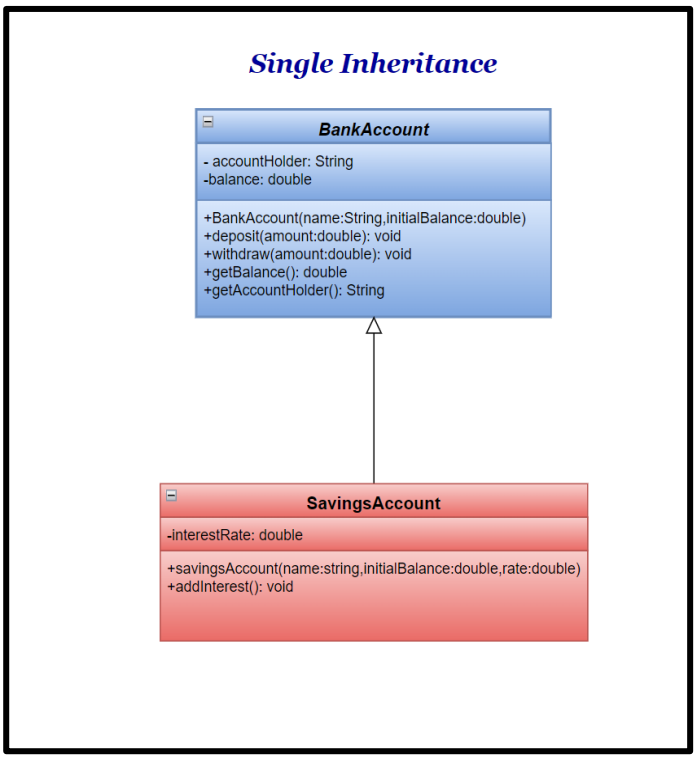
Table 2: UML Diagram Design Guidelines

Aspect	Best Practice
Class Layout	Use 3-part boxes: Name, Attributes, Methods.
Naming	Use PascalCase for class names, camelCase for attributes/methods.
Data Types	Always specify types (e.g., `int`, `string`, `void`).
Visibility Symbols	Always include `+`, `-`, or `#` for clarity.
Parameter Format	`name: type` inside method signatures.
Avoid Clutter	Show only essential attributes and methods.
Arrow Direction	Arrows point **to the base class** in inheritance.
Consistent Spacing	Keep layout neat and evenly spaced.

Task#1: Single Inheritance

Concept:

Single inheritance involves one derived class inheriting from one base class.



UML Description:

- The UML diagram includes two classes:
  - BankAccount (base class)
  - SavingsAccount (derived class)
- The arrow from SavingsAccount to BankAccount indicates inheritance.
- Shared attributes like balance and methods like deposit() are shown in BankAccount.
- SavingsAccount may add specific attributes (e.g., interestRate) or override methods.

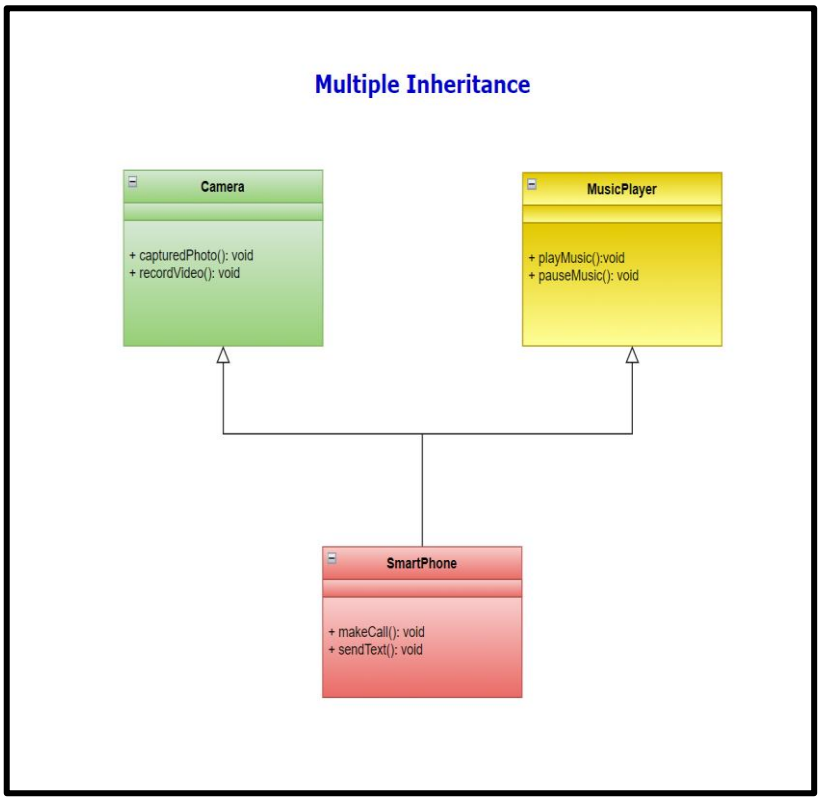
Task#2: Multiple Inheritance

Concept:

Multiple inheritance allows a class to inherit from more than one base class.

UML Description:

- Classes used in the diagram:
  - Camera and MusicPlayer (base classes)
  - SmartPhone (derived class)
- SmartPhone has two arrows pointing to Camera and MusicPlayer.
- This means it inherits functionalities such as takePhoto() and playMusic() from both sources.



How It Was Made:

- Positioned Camera and MusicPlayer side by side.
- Used two generalization arrows connecting both to SmartPhone.
- Ensured clarity by aligning arrows cleanly.

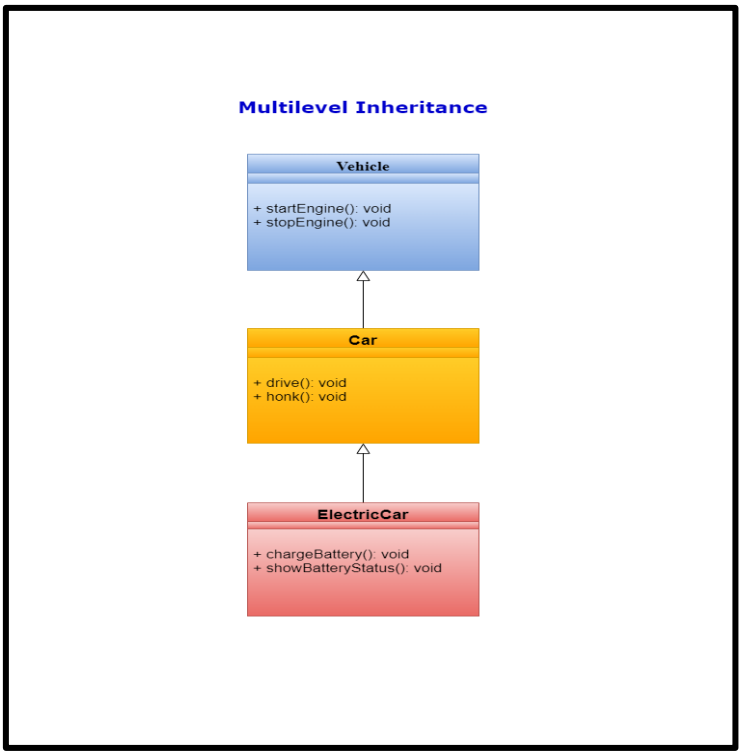
Task#3: Multilevel Inheritance

Concept:

Multilevel inheritance creates a chain of inheritance across three or more classes.

UML Description:

- The diagram consists of:
  - Vehicle (base class)
  - Car (derived from Vehicle)
  - ElectricCar (derived from Car)
- Each subclass adds or overrides functionality from its parent.



How It Was Made:

- Classes were aligned vertically to show the inheritance chain.
- One arrow from Car to Vehicle, and another from ElectricCar to Car.

Key Point: Demonstrates a parent-child-grandchild relationship.

Task#4: Hierarchical Inheritance

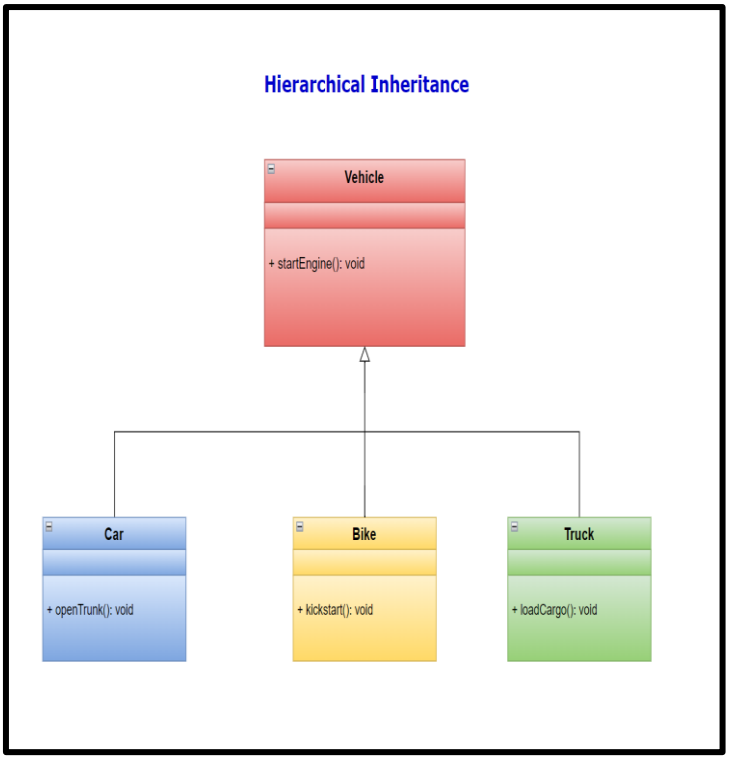
Concept:

Hierarchical inheritance occurs when multiple derived classes inherit from a single base class. It demonstrates how a parent class can provide common functionality to multiple child classes.

UML Description:

- Base Class: Vehicle
- Derived Classes: Car, Bike

- The Vehicle class includes common properties and methods such as start() or stop().
- Both Car and Bike inherit from Vehicle and may define their own specific functionalities (e.g., openTrunk() in Car, kickStart() in Bike).



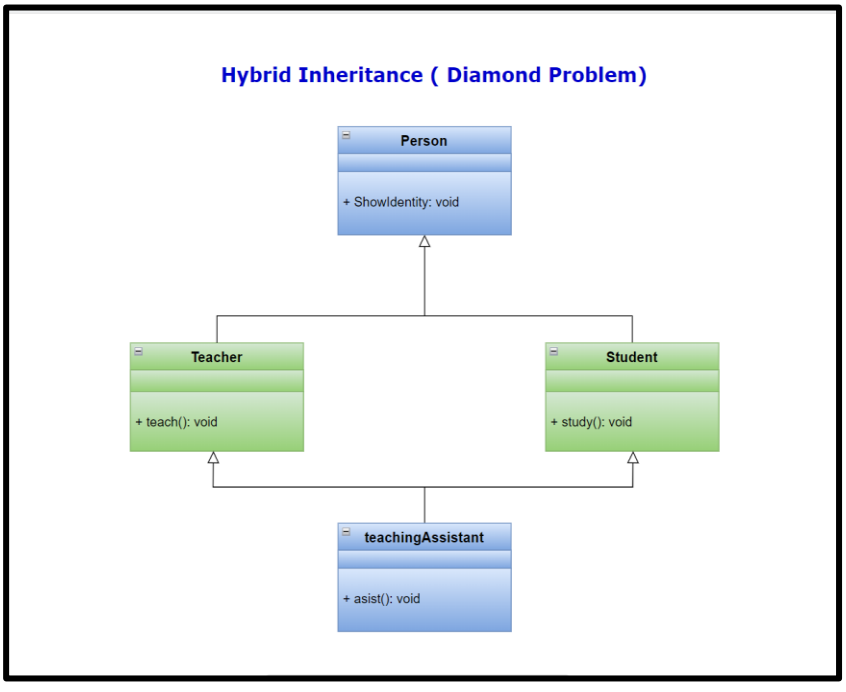
How It Was Made:

- The UML diagram places Vehicle at the top.
- Arrows point downward from Vehicle to both Car and Bike to indicate inheritance.
- Each class is represented with a box split into sections: class name, attributes, and methods.
- Shared features go into Vehicle, while class-specific behavior is added to Car and Bike.

Task#5: Hybrid Inheritance

Concept:

Hybrid inheritance is a mix of multiple, hierarchical, and multilevel inheritance.



UML Description:

- Includes a complex structure such as:
  - Person (base class)
  - Teacher and Student inheriting from Person
  - TeachingAssistant inheriting from both Teacher and Student

How It Was Made:

- Created a diamond-shaped structure to represent the issue.
- Two arrows from Teacher and Student to TeachingAssistant
- Arrows from Teacher and Student both point to Person.

Note: This leads to the **diamond problem** — ambiguity due to multiple inheritance paths to the same base class.

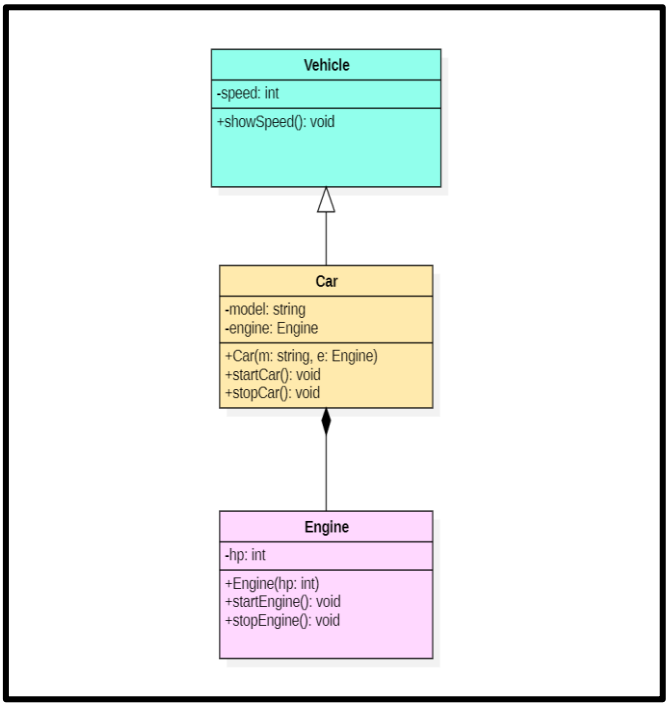
Solution (Optional):

- Use **virtual inheritance**, which can be marked in UML with notes or by labeling relationships.

Task#6: Composition

Concept:

Composition represents a strong ownership. One class (e.g., Car) contains another class (Engine), and the lifetime of the part is bound to the whole.



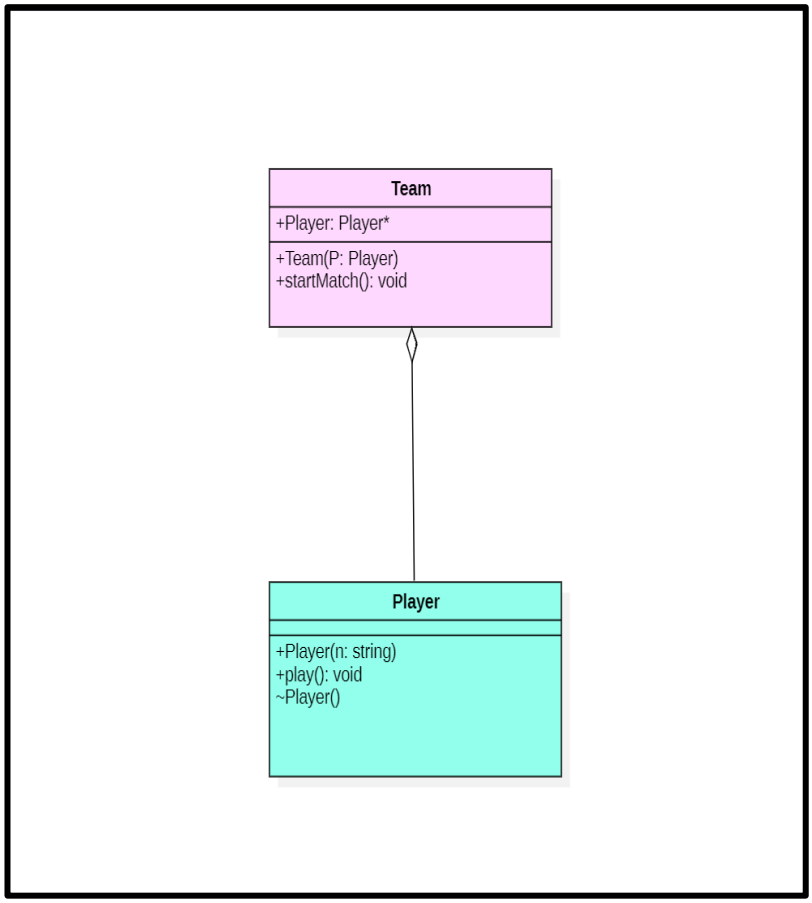
Key Points:

- Strong relationship (tight coupling)
- Contained object is destroyed with the container
- Example: Car contains an Engine

Task#6: Aggregation

Concept:

Aggregation represents a “has-a” relationship. One class (e.g., Team) contains a reference to another class (Player), but the contained object can exist independently of the container.



Key Points:

- Loose coupling between classes
- Lifespan of contained object is independent
- Example: Team has multiple Player instances