SWE 4202: OBJECT ORIENTED CONCEPTS I LAB

LAB_06: Object Relation

PREPARED BY :

FARZANA TABASSUM || LECTURER
farzana@iut-dhaka.edu

ZANNATUN NAIM SRISTY || LECTURER
zannatunnaim@iut-dhaka.edu

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ISLAMIC UNIVERSITY OF TECHNOLOGY

FEBRUARY 23, 2024

Contents

1 OBJECT RELATION

Object relationships define how objects in a system interact and collaborate. Object relation- ships can be classified into association, aggregation, and composition. Besides, inheritance, dependency, interface implementation, or realization of a contract can also be thought of as object relations.

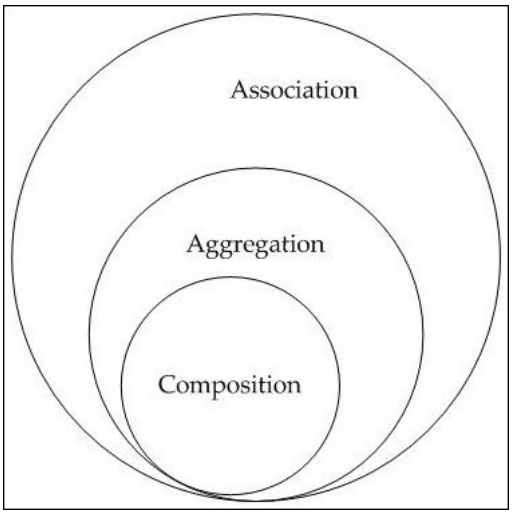

Figure 1: Association Vs. Aggregation Vs. Composition

1.1 Association

Represents a generic connection or link between two or more classes.
Characteristics: Objects of one class are related to objects of another class, often indicating some form of interaction or connection.
Example: A Teacher class associated with a Student class.

1.2 Aggregation

Represents a whole-part relationship where one class (the whole) contains another class (the part).
Characteristics: The part has a more independent lifecycle and can exist without the whole. The diamond-shaped line in UML denotes aggregation.
Example: A University class contains Departments. A Car class contains Wheel, Engine, and Seat.

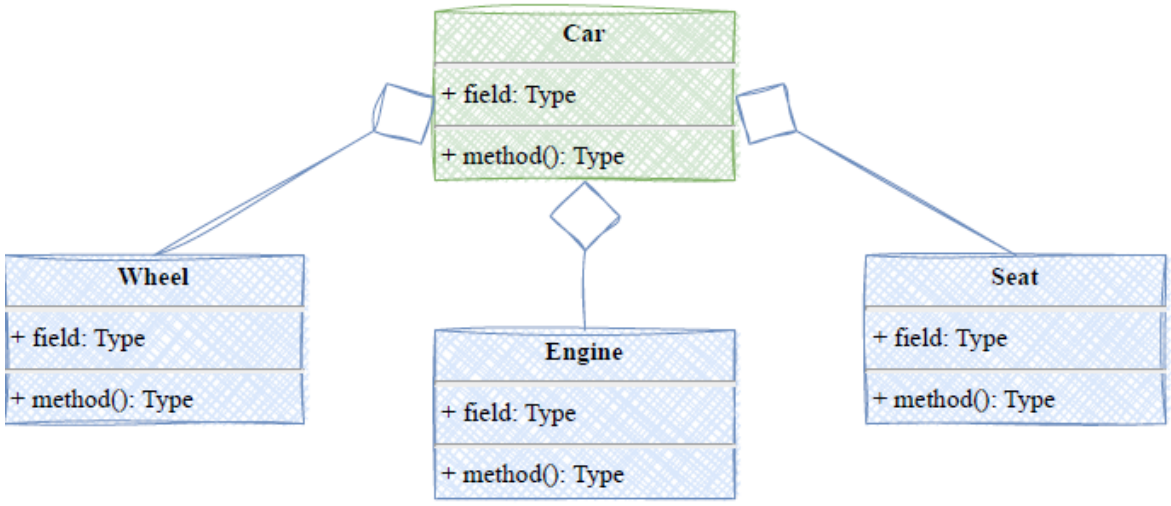

Figure 2: Association Vs. Aggregation Vs. Composition

```
    // Engine class
    class Engine {
    // Engine properties and methods

    // For simplicity, let's just details
    public String Details() {
        return "Engine details: [Type: V6, Horsepower: 300]";
        }
    }


    // Car class
    class Car {
    Engine carEngine; // Car HAS-A Engine

    // Constructor for Car class
    public Car(Engine engine) {
        this.carEngine = engine;
        }
    // Car properties and methods
    // For simplicity, we're not including any additional properties or
   methods here
    }


    // Main class
    public class Main {
    public static void main(String[] args) {
    // Creating an Engine object
    Engine engine = new Engine();

    // Creating a Car object using the constructor and passing the Engine
    object to it
    Car car = new Car(engine);

    // Accessing Car properties or methods
    // For example, printing out the details of the car's engine

    System.out.println("Car's engine details: " + car.getengine().Details
   ());
        }
    }
```

1.3 Composition

Similar to aggregation but with a stronger relationship. The part is entirely dependent on the whole. **Characteristics:** If the whole is destroyed, the parts are also destroyed. Denoted by a filled diamond in UML.
Example: A Person class composed of Brain class, Heart class, and Leg.

```
    // Brain class
    class Brain {
    // Brain properties and methods

    // For simplicity, let's just details
    public String Details() {
        return "Brain has three main regions: Cerebrum, Cerebellum,
   Brainstem";
        }
    }

```

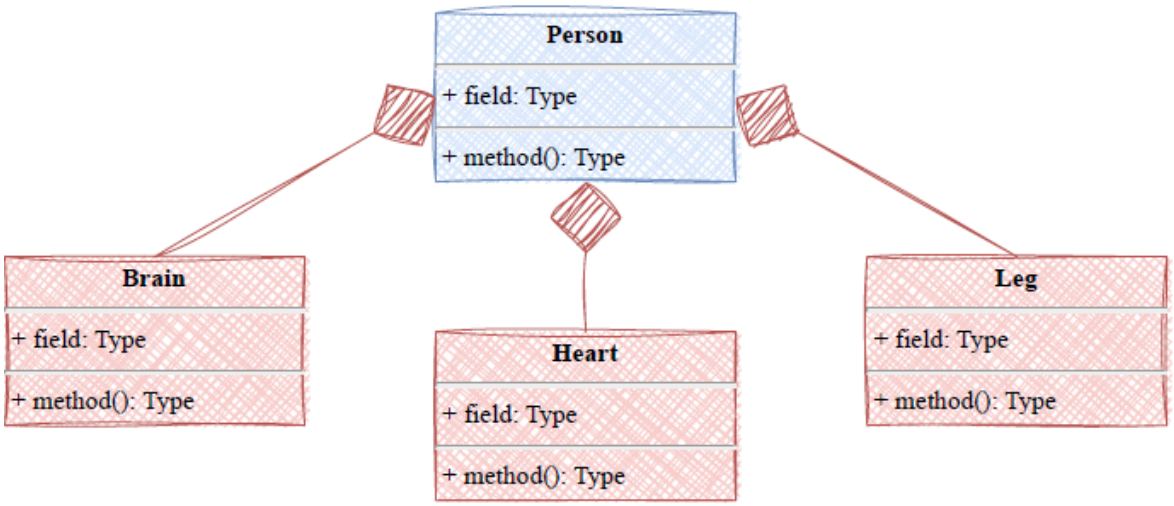

Figure 3: Association Vs. Aggregation Vs. Composition

```
11
12    // Human class
13    class Human {
14    Brain humanBrain; // Human HAS - A Brain
15
16    // Constructor for Human class
17    public Human() {
18        this.humanBrain = new Brain();
19        }
20    // Human properties and methods
21    // For simplicity, we're not including any additional properties or
      methods here
22    }
23
24
25    // Main class
26    public class Main {
27    public static void main(String[] args) {
28    // Creating a Human object
29    Human person = new Human();
30
31    // Accessing Human properties or methods
32    // For example, printing out the details of the person's brain
33
34    System.out.println("person.getHumanBrain().Details(); \}
```

2 Problem Statement 01

In a recipe-sharing platform, there are recipes, chefs, and users. Each recipe is created by one or more chefs and can be favored by multiple users. Additionally, users can follow multiple chefs. Recipes are created by chefs, and the existence of chefs is not solely dependent on any particular recipe. Similarly, users exist independently of any specific recipe or chef.
A recipe's existence is tightly bound to the chefs who create it. If a recipe is removed from the platform, the associated chefs should not be affected. However, if a chef is removed from the platform, all recipes created by that chef should also be removed.
A chef can add, or remove recipes. A user can add or remove some recipes to his or her favorite recipe list. Or do the same for his/her favorite chefs.