Name: Muhammad Fahad

Sap ID: 37125

Section: BSCS-5A

Course: Advance computer programming

Assignment#1

Instructor: Mr. Muhammad Ahmad

Github link code file: https://github.com/fahadmoon/5th-Semester/tree/java/Assignments/Assignment%201

1. Composite Relationship:

- In a composite relationship, one class contains another class as a part. The contained class cannot exist independently outside the container class. If the container class is destroyed, the contained class is also destroyed.
- It represents a "whole-part" or "has-a" relationship.
- A common example is a Car class containing Engine, Wheels, and Seats as its parts.

Main.java

```
public class Main {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.startCar();
        myCar.drive();
        myCar.park();
        myCar.occupySeat(2, "Fahad");
        myCar.occupySeat(0, "Mehboob");
        myCar.vacateSeat(2);
        myCar.occupySeat(3, "zaibi");

    }
}
```

Car.java

```
class Car {
    private Engine engine;
    private Wheel[] wheels;
    private Seat[] seats;

    public Car() {
        engine = new Engine();
        wheels = new Wheel[4];
        seats = new Seat[5];

        for (int i = 0; i < 4; i++) {
```

```
            wheels[i] = new Wheel();
        }

        for (int i = 0; i < 5; i++) {
            seats[i] = new Seat();
        }
    }

    public void startCar() {
        engine.start();
        System.out.println("Car started");
    }

    public void drive() {
        for (Wheel wheel : wheels) {
            wheel.rotate();
        }
        System.out.println("Car is moving");
    }

    public void park() {
        System.out.println("Car is parked");
    }

    public void occupySeat(int seatNumber, String personName) {
        if (seatNumber >= 0 && seatNumber < seats.length) {
            seats[seatNumber].sit(personName);
        } else {
            System.out.println("Invalid seat number");
        }
    }

    public void vacateSeat(int seatNumber) {
        if (seatNumber >= 0 && seatNumber < seats.length) {
            seats[seatNumber].getUp();
        } else {
            System.out.println("Invalid seat number");
        }
    }
}
```

Seat.java

```
class Seat {
    private String occupant;

    public Seat() {
        occupant = null;
    }

    public void sit(String personName) {
        occupant = personName;
        System.out.println(personName + " is sitting in the seat");
```

```
    }

    public void getUp() {
        if (occupant != null) {
            System.out.println(occupant + " got up from the seat");
            occupant = null;
        }
    }
}
```

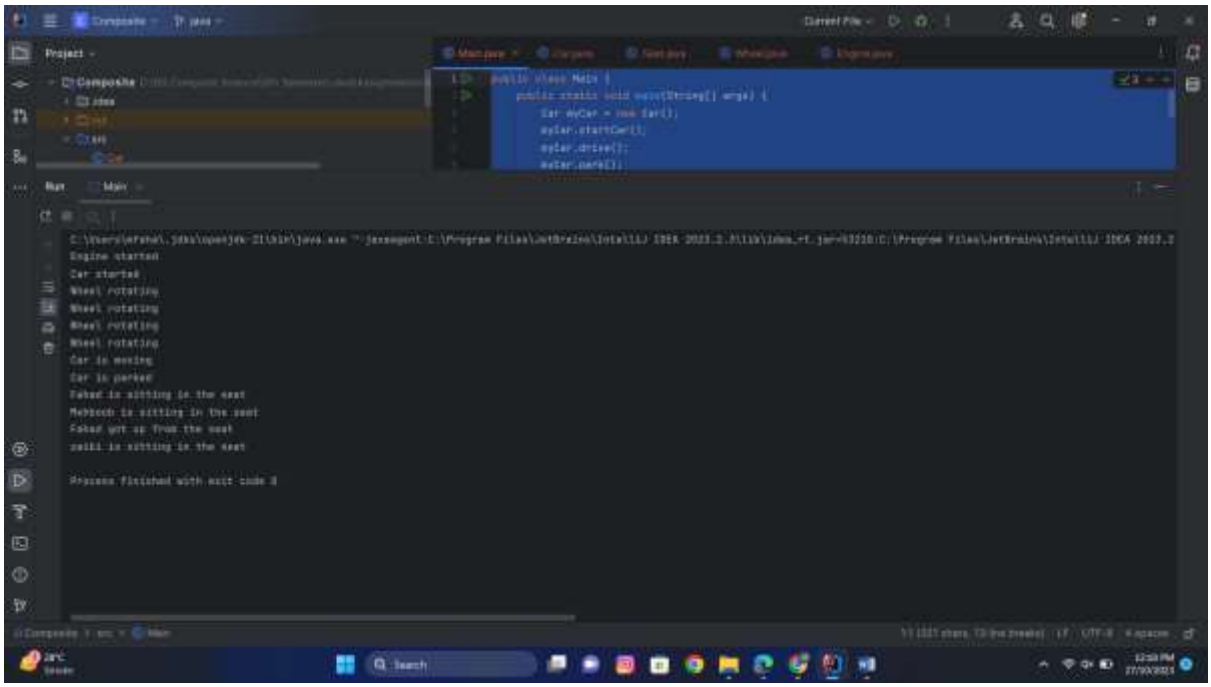
Wheal.java

```
class Wheel {
    public void rotate() {
        System.out.println("Wheel rotating");
    }
}
```

Engine.java

```
class Engine {
    public void start() {
        System.out.println("Engine started");
    }
}
```

output



2. Aggregation Relationship:

- In an aggregation relationship, one class is associated with another class, but the associated class can exist independently. It represents a "whole-part" or "has-a" relationship, just like composite, but with a key difference: the contained class has its own lifecycle and can be shared among multiple containers.
- When the container is destroyed, the contained class is not necessarily destroyed.
- A common example is a University class containing multiple Department classes.

Main.java

```
public class Main {
    public static void main(String[] args) {
        University myUniversity = new University(5);

        Department csDepartment = new Department("Computer Science");
        Department eeDepartment = new Department("Electrical Engineering");

        myUniversity.addDepartment(csDepartment);
        myUniversity.addDepartment(eeDepartment);

        myUniversity.displayDepartments();
    }
}
```

University.java

```
class University {
    private Department[] departments;
    private int departmentCount;

    public University(int maxDepartments) {
        departments = new Department[maxDepartments];
        departmentCount = 0;
    }

    public void addDepartment(Department department) {
        if (departmentCount < departments.length) {
            departments[departmentCount] = department;
            departmentCount++;
        } else {
            System.out.println("Cannot add more departments. The university is at full capacity.");
        }
    }

    public void displayDepartments() {
        System.out.println("Riphah International University Departments:");
        for (int i = 0; i < departmentCount; i++) {
```

```
            System.out.println(departments[i].getName());
        }
    }
}
```

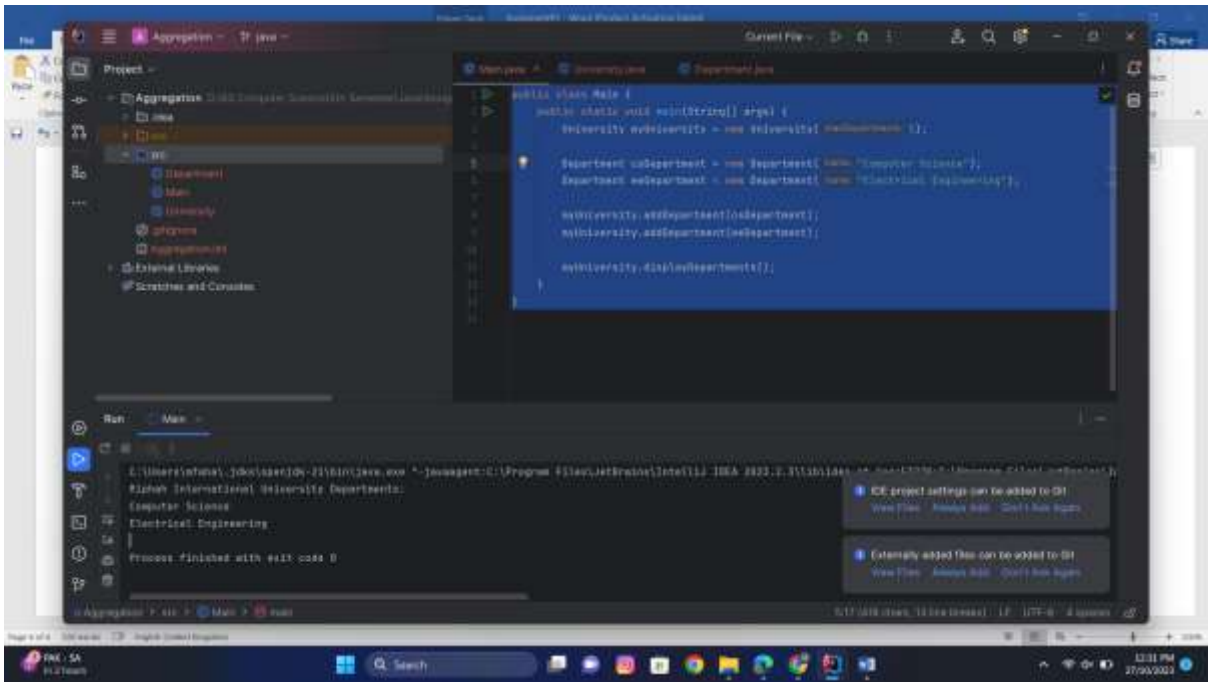
Department.java

```
class Department {
    private String name;

    public Department(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

Output:



3. Association Relationship:

- In an association relationship, two classes are related, but they are not part of each other. It represents a more general relationship where objects of one class are aware of the objects of another class.
- The relationship can be one-way or two-way. For example, a Teacher class may be associated with a Student class, but the reverse is also true. Both classes can exist independently and have their own lifecycles.
- Associations can have multiplicity, indicating how many objects of one class can be associated with objects of another class.

Main.java

```
public class Main {
    public static void main(String[] args) {
        Member member1 = new Member("Fahad");
        Member member2 = new Member("Mehboob");
        Library library = new Library();
        library.checkOutBook(member1, "Introduction to Java");
        library.checkOutBook(member2, "Programming in Python");
    }
}
```

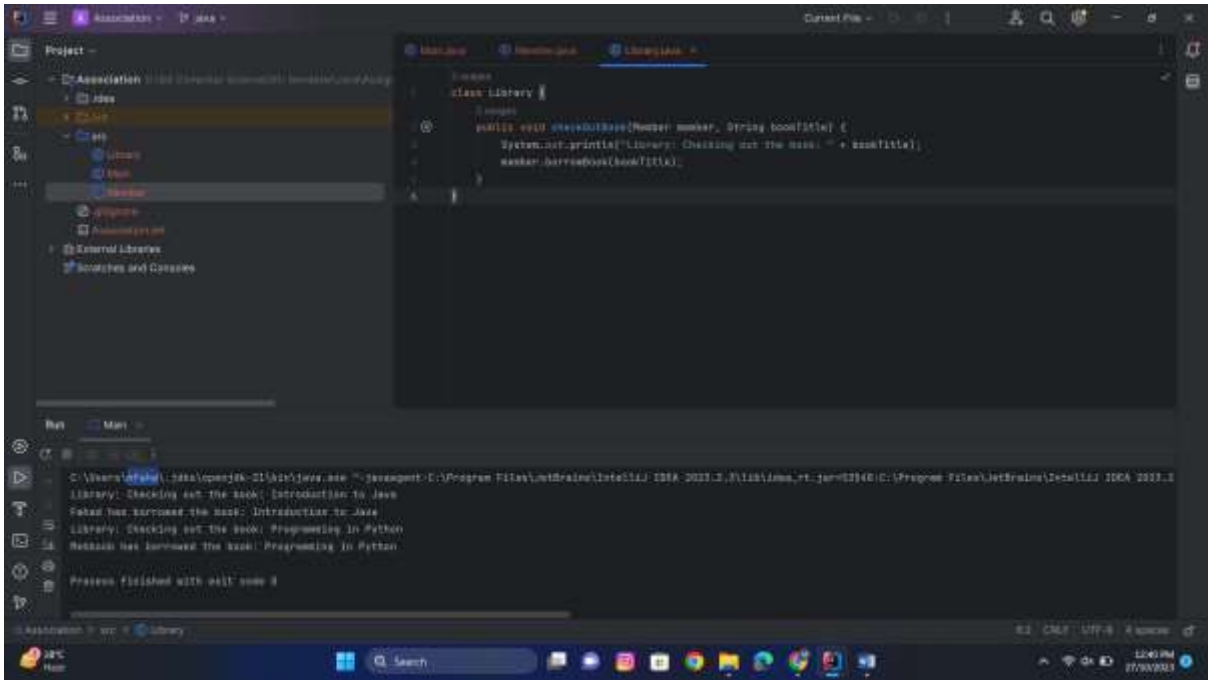
Member.java

```
class Member {
    private String name;
    public Member(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void borrowBook(String bookTitle) {
        System.out.println(name + " has borrowed the book: " + bookTitle);
    }
}
```

Library.java

```
class Library {
    public void checkOutBook(Member member, String bookTitle) {
        System.out.println("Library: Checking out the book: " + bookTitle);
        member.borrowBook(bookTitle);
    }
}
```

Output:


Project
Association
External Libraries
Scratches and Consoles
class Library {
public void checkOutBook(Member member, String bookTitle) {
System.out.println("Library: Checking out the book: " + bookTitle);
member.borrowBook(bookTitle);
}
}
Run
Main
Library: Checking out the book: Introduction to Java
Fahad has borrowed the book: Introduction to Java
Library: Checking out the book: Programming in Python
Process finished with exit code 0