SWE 4301 - OBJECT ORIENTED CONCEPTS II

LECTURE CODE EXAMPLES

Prepared by:
Md. Jubair Ibna Mostafa
Assistant Professor, IUT CSE

Department of Computer Science and Engineering
Islamic University of Technology

Contents

1 Association

Simply two or more classes perform a task. Example: Student-Teacher-Classroom, Student-Course-Enrollment. **Example**:

```java
class Teacher {
    public void teach(Student student) {
        System.out.println("Teacher is teaching.");
        student.learn();
    }
}

class Student {
    public void learn() {
        System.out.println("Student is learning.");
    }
}

class Classroom {
    private Teacher teacher;
    private Student student;

    public Classroom() {
        teacher = new Teacher();
        student = new Student();
    }

    public void startClass() {
        teacher.teach(student);
    }
}

public class Main {
    public static void main(String[] args) {
        Classroom classroom = new Classroom();
        classroom.startClass();
    }
}
```

2 Aggregation

a class uses another class to perform an activity or functionality is called Aggregation. Example: University-Department, Department-Student, Company-Employee. **Example**:

```
class Employee {
    String name;
    String id;
    void work(){
        System.out.println("Doing my job");
    }
}

class Department {
    private List<Employee> employees;

    public Department() {
        employees = new ArrayList<>();
    }

    public void addEmployee(Employee employee) {
        employees.add(employee);
    }
}
```

3 Composition

Composition is the strict relation. Normally, it describes as Whole-Part Relationship. Example: House-Room, Human-Heart, Car-Engine. **Example 1**:

```
class Engine {
    String model;
    int capacity;
}

class Wheel {
    String color;
}

class Car {
    private Engine engine;
    private Wheel[] wheels;

    public Car() {
        engine = new Engine();
        wheels = new Wheel[4];
        for (int i = 0; i < 4; i++) {
            wheels[i] = new Wheel();
        }
    }

    // Car-specific methods
}
```

Example 2: Here TextEdior-File has an aggregation Relation and TextEditor-Buffer has Composition relation.

```
class File {
    private String fileName;

    public File(String fileName) {
        this.fileName = fileName;
    }

    public String getFileName() {
        return fileName;
    }
```

```java
}

class Buffer {
    private String content;

    public Buffer(String content) {
        this.content = content;
    }

    public String getContent() {
        return content;
    }
}

class TextEditor {
    private Buffer buffer;
    private File file;

    public TextEditor(File file) {
        this.file = file;
        this.buffer = new Buffer("");
    }

    public void open() {
        System.out.println("Opening file: " + file.getFileName());
        // Load file content into buffer
        buffer = new Buffer("Content of " + file.getFileName());
    }

    public void edit(String newContent) {
        System.out.println("Editing content...");
        buffer = new Buffer(newContent);
    }

    public void save() {
        System.out.println("Saving changes to file: " + file.getFileName());
        // Save buffer content to file
    }

    public void close() {
        System.out.println("Closing file: " + file.getFileName());
        buffer = null; // Buffer is destroyed as part of composition
    }
}

public class Main {
    public static void main(String[] args) {
        File file = new File("document.txt");
        TextEditor textEditor = new TextEditor(file);

        textEditor.open();
        System.out.println("Initial Content: " + textEditor.getContent());

        textEditor.edit("Updated content.");
        System.out.println("Updated Content: " + textEditor.getContent());

        textEditor.save();
        textEditor.close();
    }
}
```

4 Delegation

Delegation means giving away a task to other classes without performing it. The class that passes the work to some other class is called Delegator, and the class that performs the task is called Delegatee.

Bad Example:

```
class Train {
    public void bookTicket(Passenger passenger) {
        System.out.println("Train ticket booked for " + passenger.getName());
    }
}

class Plane {
    public void bookTicket(Passenger passenger) {
        System.out.println("Plane ticket booked for " + passenger.getName());
    }
}

class Passenger {
    private String name;

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

    public String getName() {
        return name;
    }

    public void bookTicketForTravel(String mode) {
        if (mode.equals("train")) {
            Train train = new Train();
            train.bookTicket(this);
        } else if (mode.equals("plane")) {
            Plane plane = new Plane();
            plane.bookTicket(this);
        } else {
            System.out.println("Invalid travel mode");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Passenger passenger1 = new Passenger("Alice");
        Passenger passenger2 = new Passenger("Bob");

        passenger1.bookTicketForTravel("train");
        passenger2.bookTicketForTravel("plane");
        passenger1.bookTicketForTravel("bus"); // Invalid mode
    }
}
```

Example using Polymorphism:

```
// Passenger class represents a passenger with a name and ID
class Passenger {
    private int id;
    private String name;

    public Passenger(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }
}

```

```java
// Ticket interface for booking tickets
interface Ticket {
    boolean bookTicket(Passenger passenger);
}

// Train class that implements Ticket interface
class Train implements Ticket {
    private int trainNumber;
    private int availableSeats;

    public Train(int trainNumber, int availableSeats) {
        this.trainNumber = trainNumber;
        this.availableSeats = availableSeats;
    }

    public boolean bookTicket(Passenger passenger) {
        if (availableSeats > 0) {
            availableSeats--;
            System.out.println("Ticket booked for passenger " +
   passenger.getName() + " on train " + trainNumber);
            return true;
        } else {
            System.out.println("No available seats on train " +
   trainNumber);
            return false;
        }
    }
}

// Plane class that implements Ticket interface
class Plane implements Ticket {
    private int flightNumber;
    private int availableSeats;

    public Plane(int flightNumber, int availableSeats) {
        this.flightNumber = flightNumber;
        this.availableSeats = availableSeats;
    }

    public boolean bookTicket(Passenger passenger) {
        if (availableSeats > 0) {
            availableSeats--;
            System.out.println("Ticket booked for passenger " +
    passenger.getName() + " on flight " + flightNumber);
            return true;
        } else {
            System.out.println("No available seats on flight " +
    flightNumber);
            return false;
        }
    }
}

// BookingManagement class delegates booking to Train or Plane objects
class BookingManagement {
    public static void bookTicket(Ticket ticket, Passenger passenger) {
        ticket.bookTicket(passenger);
    }
}

// Main class to demonstrate the ticket booking system
public class Main {
    public static void main(String[] args) {
        Passenger passenger1 = new Passenger(1, "Alice");
        Passenger passenger2 = new Passenger(2, "Bob");

        Train train = new Train(123, 50);
        Plane plane = new Plane(456, 100);

        BookingManagement.bookTicket(train, passenger1);
        BookingManagement.bookTicket(plane, passenger2);
    }
}
```