Object Modelling: Object Relationships and Communication

Problem 1: Library and Books (Aggregation)

Description:

Create a Library class that contains multiple Book objects. Model the relationship such that a library can have many books, but a book can exist independently (outside of a specific library).

Tasks:

- Define a Library class with an ArrayList of Book objects.
- Define a Book class with attributes such as title and author.
- Demonstrate the aggregation relationship by creating books and adding them to different libraries.

Goal:

Understand aggregation by modeling a real-world relationship where the Library aggregates Book objects.

Object Diagram:

library1 : Library

└── books →

├── book1 : Book ["1984", "George Orwell"]

└── book2 : Book ["Brave New World", "Aldous Huxley"]

Problem 2: Bank and Account Holders (Association)

Description:

Model a relationship where a Bank has Customer objects associated with it. A Customer can have multiple bank accounts, and each account is linked to a Bank.

Tasks:

- Define a Bank class and a Customer class.
- Use an association relationship to show that each customer has an account in a bank.
- Implement methods that enable communication, such as openAccount() in the Bank class and viewBalance() in the Customer class.

Goal:

Illustrate association by setting up a relationship between customers and the bank.

Object Diagram:

bank1 : Bank ["SBI"]

↑ ↑

| |

customer1 customer2

: Customer : Customer

["Ayush"] ["Riya"]

Problem 3: Company and Departments (Composition)

Description:

A Company has several Department objects, and each department contains Employee objects. Model this using composition, where deleting a company should also delete all departments and employees.

Tasks:

- Define a Company class that contains multiple Department objects.
- Define an Employee class within each Department.
- Show the composition relationship by ensuring that when a Company object is deleted, all associated Department and Employee objects are also removed.

Goal:

Understand composition by implementing a relationship where Department and Employee objects cannot exist without a Company.

Object Diagram:

company1 : Company ["TechCorp"]

└── departments →

├── dept1 : Department ["HR"]

│ └── employees →

│ └── emp1 : Employee ["John"]

└── dept2 : Department ["IT"]

└── employees →

├── emp2 : Employee ["Alice"]

└── emp3 : Employee ["Bob"]

Problem 4: School and Students with Courses (Aggregation and Association)

Description:

Model a School with multiple Student objects, where each student can enroll in multiple courses, and each course can have multiple students.

Tasks:

- Define School, Student, and Course classes.
- Model an association between Student and Course to show that students can enroll in multiple courses.
- Model an aggregation relationship between School and Student.
- Demonstrate how a student can view the courses they are enrolled in and how a course can show its enrolled students.

Goal:

Practice association by modeling many-to-many relationships between students and courses.

Object Diagram:

school1 : School

└── students →

├── stu1 : Student ["Amit"]

│ └── courses →

│ ├── course1 : Course ["Maths"]

│ └── course2 : Course ["Science"]

└── stu2 : Student ["Neha"]

└── courses →

└── course2 : Course ["Science"]

Problem 5: University with Faculties and Departments (Composition and Aggregation)

Description:

Create a University with multiple Faculty members and Department objects. Model it so that the University and its Departments are in a composition relationship (deleting a university deletes all departments), and the Faculty members are in an aggregation relationship (faculty can exist outside of any specific department).

Tasks:

- Define a University class with Department and Faculty classes.
- Demonstrate how deleting a University also deletes its Departments.
- Show that Faculty members can exist independently of a Department.

Goal:

Understand the differences between composition and aggregation in modelling complex hierarchical relationships.

Object Diagram:

university1 : University

├── departments →

│ └── dept1 : Department ["CSE"]

└── faculties →

├── faculty1 : Faculty ["Dr. Sharma"]

└── faculty2 : Faculty ["Dr. Khan"]

Problem 6: Hospital, Doctors, and Patients (Association and Communication)

Description:

Model a Hospital where Doctor and Patient objects interact through consultations. A doctor can see multiple patients, and each patient can consult multiple doctors.

Tasks:

- Define a Hospital class containing Doctor and Patient classes.
- Create a method consult() in the Doctor class to show communication, which would display the consultation between a doctor and a patient.
- Model an association between doctors and patients to show that doctors and patients can have multiple relationships.

Goal:

Practice creating an association with communication between objects by modeling doctor-patient consultations.

Object Diagram:

hospital1 : Hospital

```
├── doctors →
│     ├── doc1 : Doctor ["Dr. Meena"]
│     └── doc2 : Doctor ["Dr. Ravi"]
└── patients →
      ├── pat1 : Patient ["Rohit"]
      └── pat2 : Patient ["Pooja"]
```

Consultations:

doc1 → pat1

doc2 → pat1, pat2

Problem 7: E-commerce Platform (Customer, Orders, Products)

Description:

Design an e-commerce platform with Order, Customer, and Product classes. Model relationships where a Customer places an Order, and each Order contains multiple Product objects.

Goal:

Show communication and object relationships by designing a system where customers communicate through orders, and orders aggregate products.

Object Diagram:

customer1 : Customer ["Ayush"]

```
└── orders →
      └── order1 : Order
            └── products →
                  ├── prod1 : Product ["Laptop"]
                  └── prod2 : Product ["Mouse"]
```

Problem 8: University Management System

Description:

Model a university system with Student, Professor, and Course classes. Students enroll in courses, and professors teach courses. Ensure students and professors can communicate through methods like enrollCourse() and assignProfessor().

Goal:

Use association and aggregation to create a university system that emphasizes relationships and interactions among students, professors, and courses.

Object Diagram: