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.

```
+----------------+      <>-----------+
|    Library     | 1    * |    Book   |
+----------------+----------+----------+
| - libraryName  |          | - title :String |
| - books: List  |        | - author: String |
+----------------+        +----------+
| +addBook()     |   | +displayBook() |
| +displayLibrary()|      +----------+
+----------------+
Aggregation: "Library has-a Book", but
Book can exist independently.
```

```
                    +----------------+
                    |    Library     |
                    +----------------+
                   | libraryName    |
                    | books: List<>  |
                    +----------------+
                            |
               +------------+------------+
                      |                 |
                 +--------+        +------------+
   | Book 1      |  Book 2  |   <--- Book exists independently
                 +--------+        +------------+
```

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.

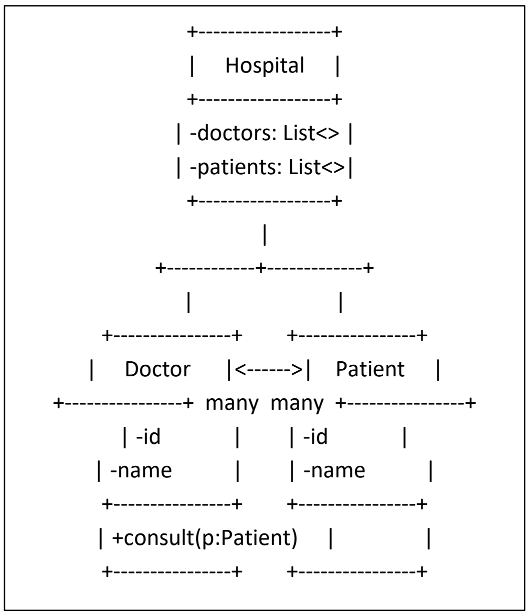
```
          +---------+      uses      +------------+
        |  Bank   |<----------------->|  Customer  |
         +---------+    associates    +------------+

    | -name :String  |              | -name :String     |
    | -list: ArrayList   |                | -id   :int     |
     +------------------------+         | -balance   |
     | +openAccount()              | +viewBalance() |
       | +showAllCustomers()        | +deposit()     |
          +---------+              +--------------+


                Relationship: Association
            - A Bank knows about its Customers.
      - A Customer is associated with a Bank but exists
```

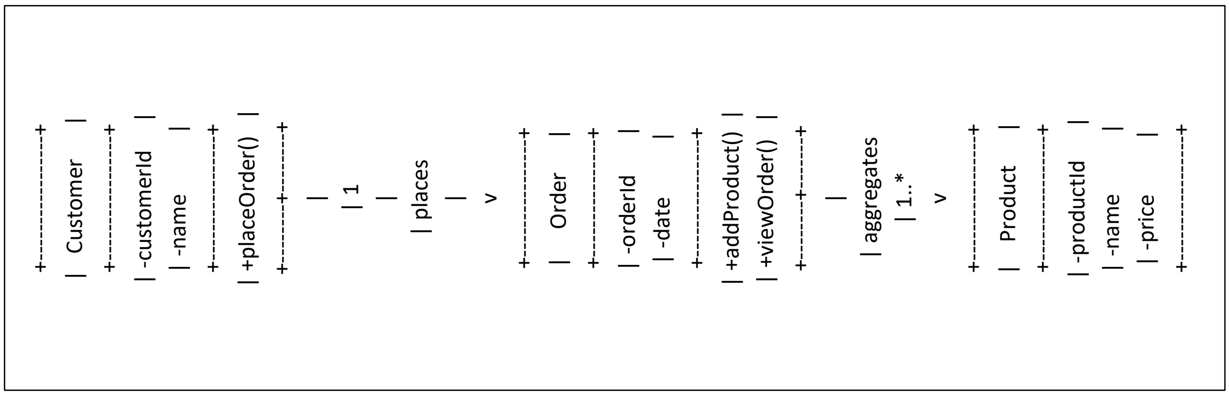
Problem 3: Hospital, Doctors, and Patients (Association and Communication)

- **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.



Problem 4: E-commerce Platform with Orders, Customers, and 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.



Problem 5: 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.

```
+------------------+      +------------------+
|    Student     |       |    Professor    |
+------------------+      +------------------+
| -studentId      |       | -professorId    |
| -name           |       | -name           |
+------------------+      +------------------+
| +enrollCourse()  |       | +assignCourse()  |
+---------+--------+      +---------+--------+
        \                 /
         \               /
          \  enrolls / teaches
           \           /
            v         v
        +---------------------+
        |       Course        |
        +---------------------+
        | -courseId           |
        | -title              |
        +---------------------+
        | +addStudent()       |
        | +setProfessor()     |
        +---------------------+
```