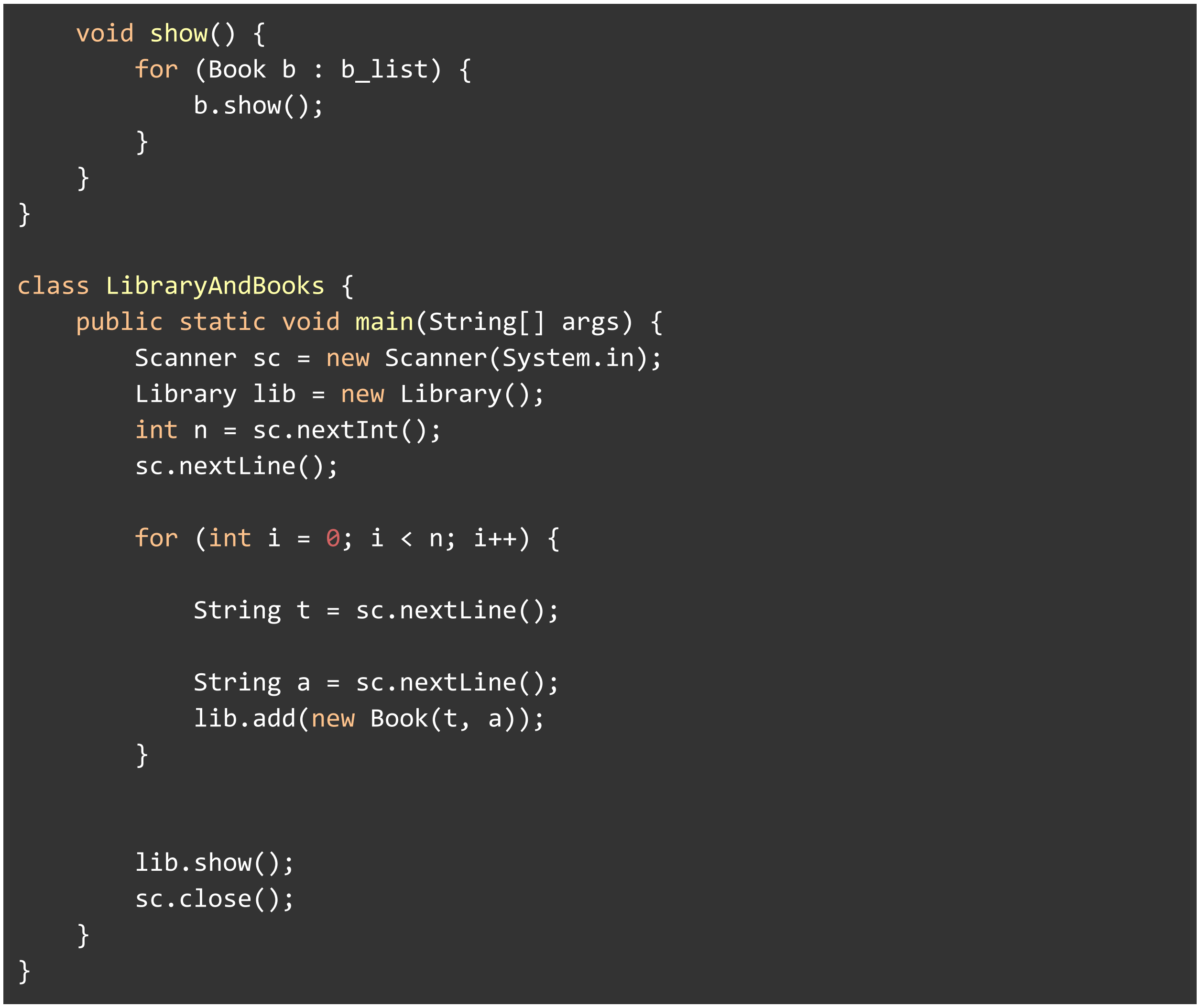
## Object modeling: Object relationships and Communication,

## Assisted Problems

### 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.

```java
import java.util.*;

class Book {
    String t, a;

    Book(String t, String a) {
        this.t = t;
        this.a = a;
    }

    void show() {
        System.out.println(t + " by " + a);
    }
}

class Library {
    List<Book> b_list = new ArrayList<>();

    void add(Book b) {
        b_list.add(b);
    }
```



```
    void show() {
        for (Book b : b_list) {
            b.show();
        }
    }
}

class LibraryAndBooks {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Library lib = new Library();
        int n = sc.nextInt();
        sc.nextLine();

        for (int i = 0; i < n; i++) {

            String t = sc.nextLine();

            String a = sc.nextLine();
            lib.add(new Book(t, a));
        }


        lib.show();
        sc.close();
    }
}
```

**Problem 2: Bank and Account Holders (Association)**

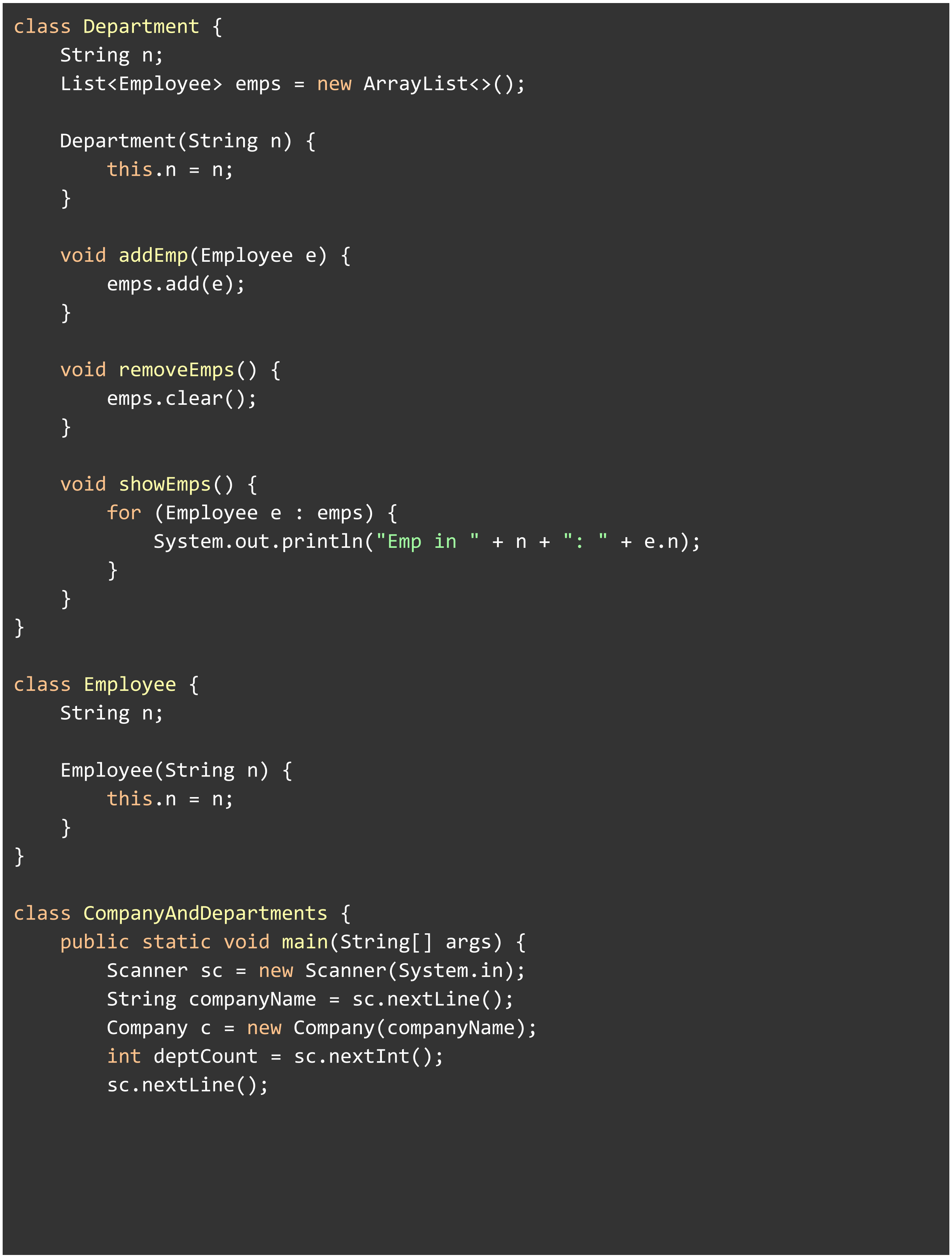
- **Description**: Model a relationship where a `Bank` has `Customer` objects associated with it. A `Customer` can have multiple bank accounts, and each account is linked to a `Bank`.
- **Tasks**:
    - Define a `Bank` class and a `Customer` class.
    - Use an association relationship to show that each 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.

```
import java.util.*;

class Account {
    int acc_no;
    double bal;

    Account(int acc_no, double bal) {
        this.acc_no = acc_no;
        this.bal = bal;
    }
}

class Customer {
    String name;
    List<Account> acc_list = new ArrayList<>();

    Customer(String name) {
        this.name = name;
    }
}








class Bank {
    String name;
    List<Customer> c_list = new ArrayList<>();

    Bank(String name) {
        this.name = name;
    }

    void openAccount(Customer c, Account a) {
        c.acc_list.add(a);
        if (!c_list.contains(c))
            c_list.add(c);
    }
}

class BankAndAccountHolders{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Bank bank = new Bank(sc.nextLine());
        int n = sc.nextInt();
        sc.nextLine();

        for (int i = 0; i < n; i++) {
            Customer cust = new Customer(sc.nextLine());
            int m = sc.nextInt();

            for (int j = 0; j < m; j++)
                bank.openAccount(cust, new Account(sc.nextInt(),
sc.nextDouble()));

            sc.nextLine();
        }

        for (Customer c : bank.c_list) {
            System.out.println(c.name + "'s Accounts:");
            for (Account acc : c.acc_list)
                System.out.println("Account No: " + acc.acc_no + ", Balance: ₹"
+ acc.bal);
        }

        sc.close();
    }
}
```

### 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.

```
import java.util.*;

class Company {
    String n;
    List<Department> dpts = new ArrayList<>();

    Company(String n) {
        this.n = n;
    }

    void addDept(Department d) {
        dpts.add(d);
    }

    void removeDepts() {
        dpts.clear();
    }

    void delCompany() {
        System.out.println("Deleting company: " + n);
        removeDepts();
    }
}
```



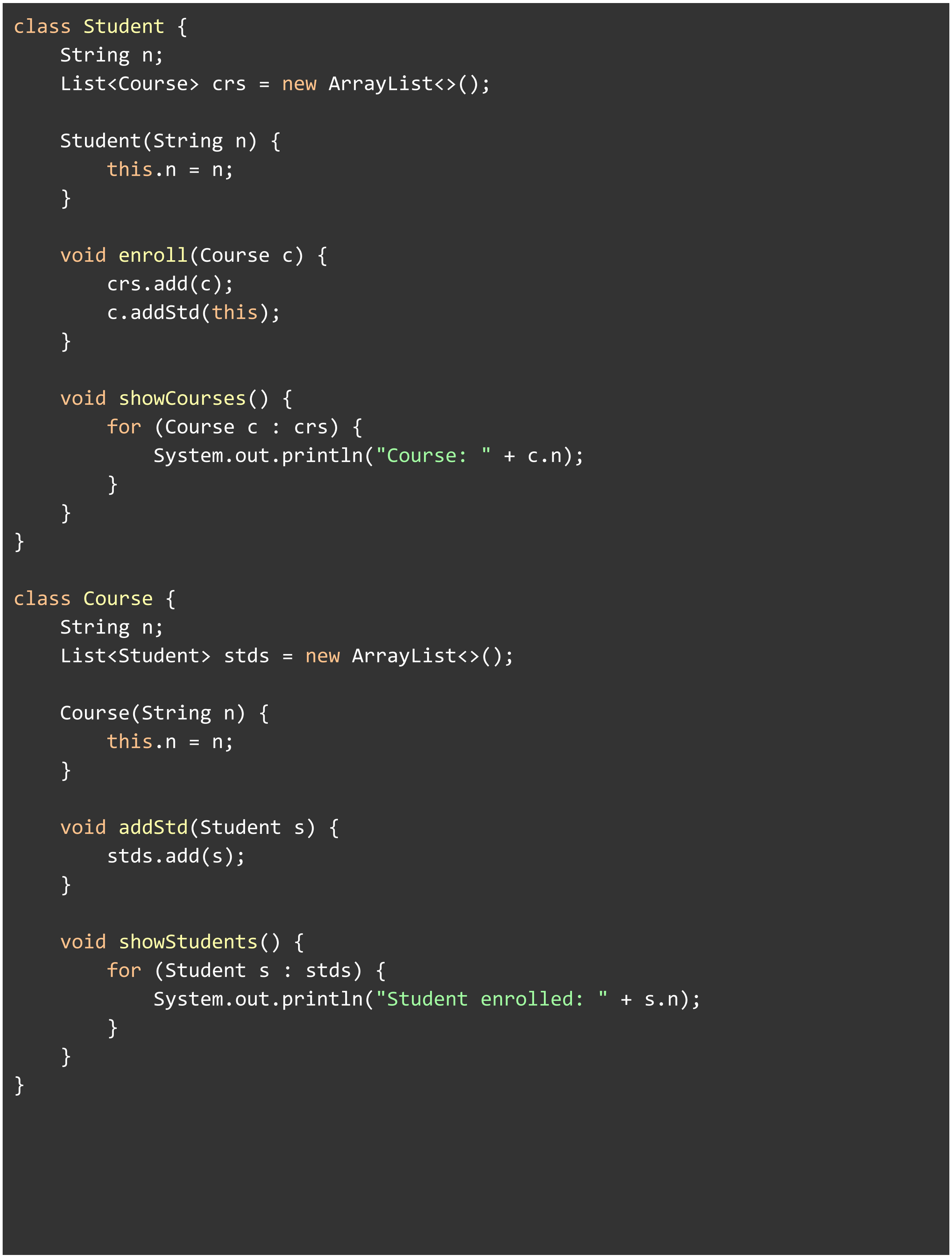
```java
class Department {
    String n;
    List<Employee> emps = new ArrayList<>();

    Department(String n) {
        this.n = n;
    }

    void addEmp(Employee e) {
        emps.add(e);
    }

    void removeEmps() {
        emps.clear();
    }

    void showEmps() {
        for (Employee e : emps) {
            System.out.println("Emp in " + n + ": " + e.n);
        }
    }
}

class Employee {
    String n;

    Employee(String n) {
        this.n = n;
    }
}

class CompanyAndDepartments {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String companyName = sc.nextLine();
        Company c = new Company(companyName);
        int deptCount = sc.nextInt();
        sc.nextLine();
        for (int i = 0; i < deptCount; i++) {
            String deptName = sc.nextLine();
            Department dept = new Department(deptName);

            System.out.print("Enter number of employees in " + deptName + ":
");
            int empCount = sc.nextInt();
            sc.nextLine();

            for (int j = 0; j < empCount; j++) {
                String empName = sc.nextLine();
                Employee emp = new Employee(empName);
                dept.addEmp(emp);
            }

            c.addDept(dept);
        }

        System.out.println("\nBefore company deletion:");
        for (Department dept : c.dpts) {
            dept.showEmps();
        }

        c.delCompany();

        System.out.println("\nAfter company deletion:");
        for (Department dept : c.dpts) {
            dept.showEmps();
        }

        sc.close();
    }
}
```

## Self Problems

### Problem 1: School and Students with Courses (Association and Aggregation)

- **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.

```java
import java.util.*;

class School {
    String n;
    List<Student> stds = new ArrayList<>();

    School(String n) {
        this.n = n;
    }

    void addStd(Student s) {
        stds.add(s);
    }

    void showStudents() {
        for (Student s : stds) {
            System.out.println("Student: " + s.n);
        }
    }
}
```



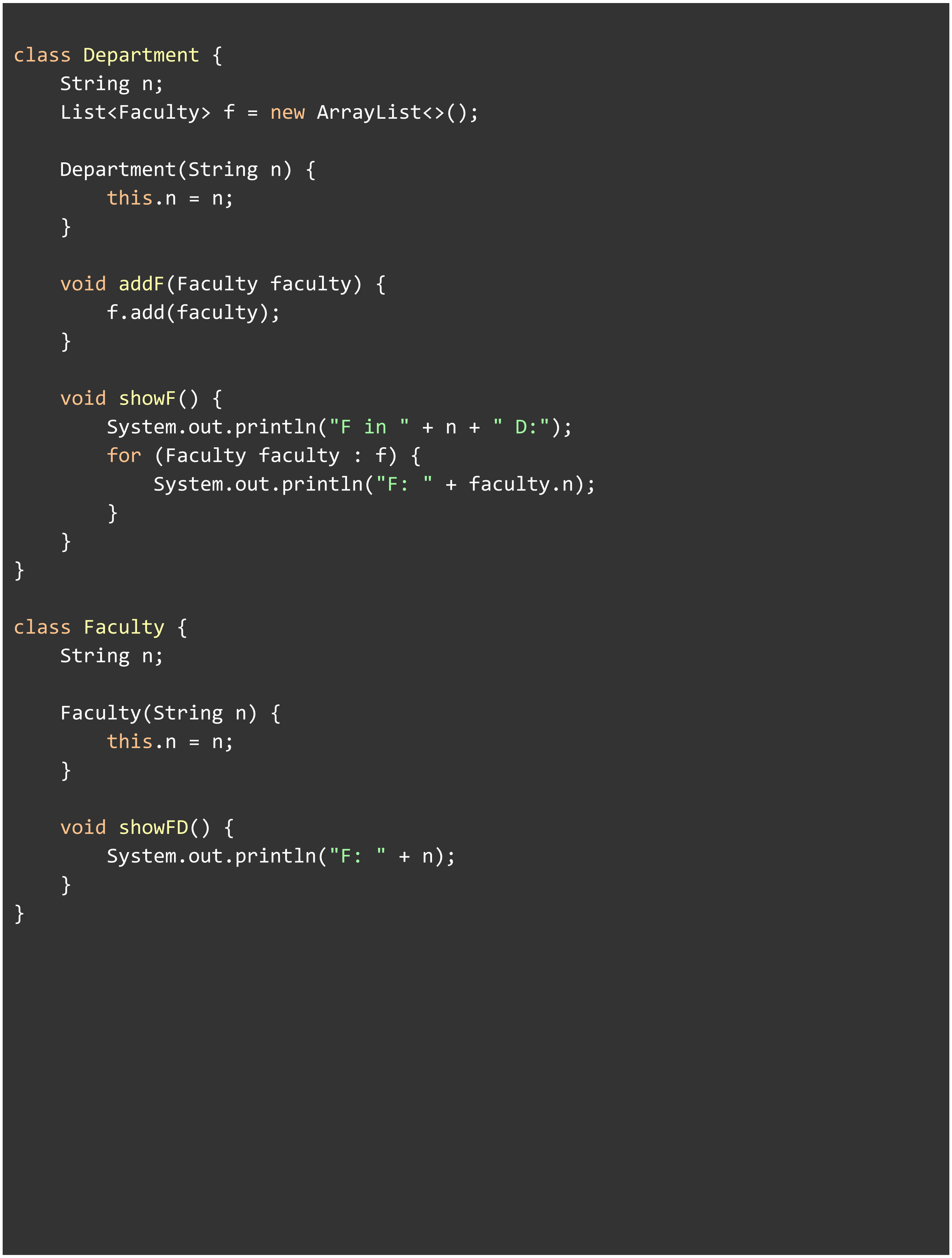
```java
class Student {
    String n;
    List<Course> crs = new ArrayList<>();

    Student(String n) {
        this.n = n;
    }

    void enroll(Course c) {
        crs.add(c);
        c.addStd(this);
    }

    void showCourses() {
        for (Course c : crs) {
            System.out.println("Course: " + c.n);
        }
    }
}

class Course {
    String n;
    List<Student> stds = new ArrayList<>();

    Course(String n) {
        this.n = n;
    }

    void addStd(Student s) {
        stds.add(s);
    }

    void showStudents() {
        for (Student s : stds) {
            System.out.println("Student enrolled: " + s.n);
        }
    }
}
```

```
class SchoolAndStudents {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        String schoolName = sc.nextLine();
        School sch = new School(schoolName);

        int stdCount = sc.nextInt();
        sc.nextLine();

        for (int i = 0; i < stdCount; i++) {

            String studentName = sc.nextLine();
            Student std = new Student(studentName);

            System.out.print("Enter number of courses for " + studentName + ":
");
            int courseCount = sc.nextInt();
            sc.nextLine();

            for (int j = 0; j < courseCount; j++) {

                String courseName = sc.nextLine();
                Course crs = new Course(courseName);
                std.enroll(crs);
            }

            sch.addStd(std);
        }

        System.out.println("\nSchool Students:");
        sch.showStudents();

        System.out.println("\nCourses and Enrolled Students:");
        for (Student std : sch.stds) {
            std.showCourses();
        }

        sc.close();
    }
}
```

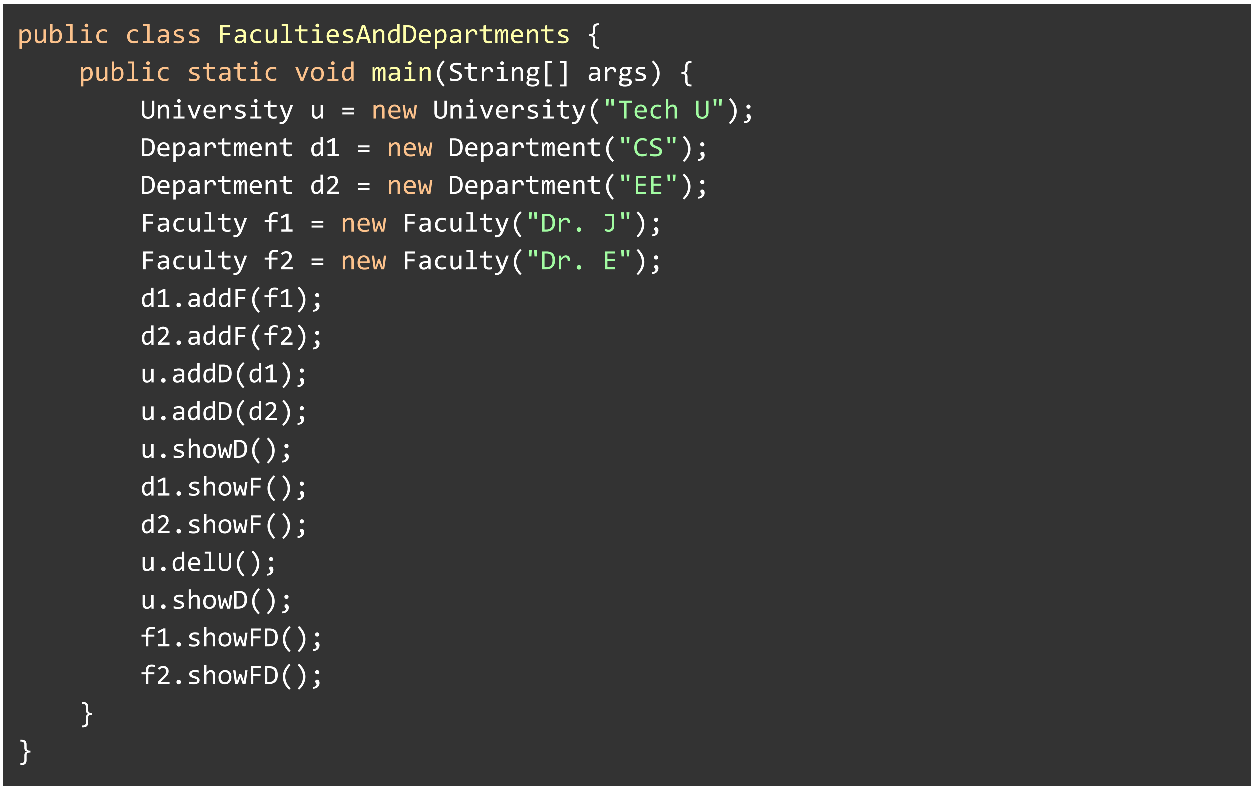
### Problem 2: University with Faculties and Departments (Composition and Aggregation)

- **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 modeling complex hierarchical relationships.

```java
import java.util.*;

class University {
    String n;
    List<Department> d = new ArrayList<>();

    University(String n) {
        this.n = n;
    }

    void addD(Department dept) {
        d.add(dept);
    }

    void remD(Department dept) {
        d.remove(dept);
    }

    void showD() {
        System.out.println("D in " + n + ":");
        for (Department dept : d) {
            System.out.println("D: " + dept.n);
        }
    }

    void delU() {
        d.clear();
    }
}
```



```java
class Department {
    String n;
    List<Faculty> f = new ArrayList<>();

    Department(String n) {
        this.n = n;
    }

    void addF(Faculty faculty) {
        f.add(faculty);
    }

    void showF() {
        System.out.println("F in " + n + " D:");
        for (Faculty faculty : f) {
            System.out.println("F: " + faculty.n);
        }
    }
}

class Faculty {
    String n;

    Faculty(String n) {
        this.n = n;
    }

    void showFD() {
        System.out.println("F: " + n);
    }
}
```

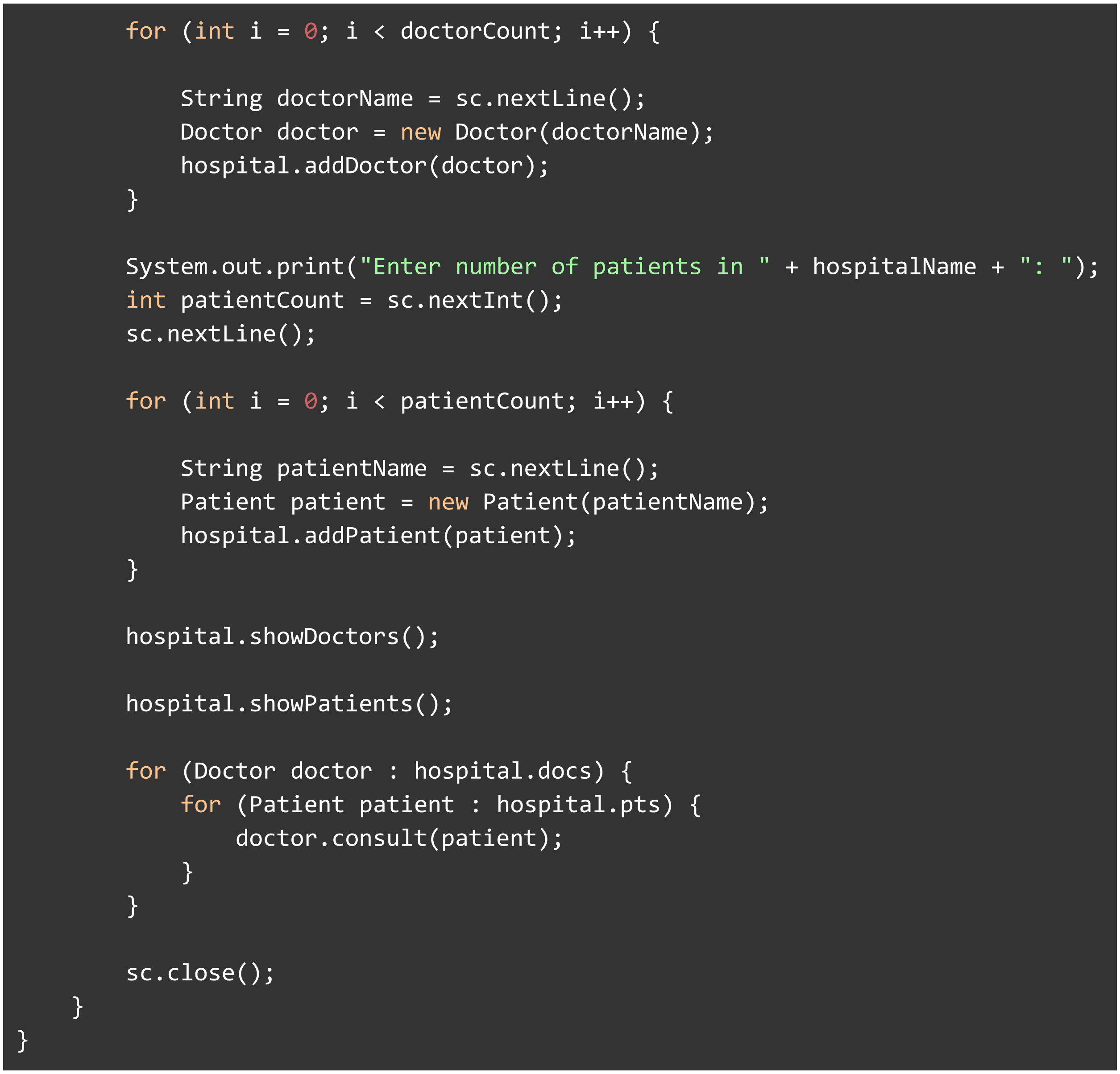


```
public class FacultiesAndDepartments {
    public static void main(String[] args) {
        University u = new University("Tech U");
        Department d1 = new Department("CS");
        Department d2 = new Department("EE");
        Faculty f1 = new Faculty("Dr. J");
        Faculty f2 = new Faculty("Dr. E");
        d1.addF(f1);
        d2.addF(f2);
        u.addD(d1);
        u.addD(d2);
        u.showD();
        d1.showF();
        d2.showF();
        u.delU();
        u.showD();
        f1.showFD();
        f2.showFD();
    }
}
```

### 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.

```
import java.util.*;

class Hospital {
    String n;
    List<Doctor> docs = new ArrayList<>();
    List<Patient> pts = new ArrayList<>();

    Hospital(String n) {
        this.n = n;
    }

    void addDoctor(Doctor d) {
        docs.add(d);
    }

    void addPatient(Patient p) {
        pts.add(p);
    }

    void showDoctors() {
        for (Doctor d : docs) {
            System.out.println("Doctor: " + d.n);
        }
    }
    void showPatients() {
        for (Patient p : pts) {
            System.out.println("Patient: " + p.n);
        }
    }
}

class Doctor {
    String n;

    Doctor(String n) {
        this.n = n;
    }

    void consult(Patient p) {
        System.out.println(n + " is consulting " + p.n);
    }
}

class Patient {
    String n;

    Patient(String n) {
        this.n = n;
    }
}

class HospitalDoctorPatient {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        String hospitalName = sc.nextLine();
        Hospital hospital = new Hospital(hospitalName);

        System.out.print("Enter number of doctors in " + hospitalName + ": ");
        int doctorCount = sc.nextInt();
        sc.nextLine();
```

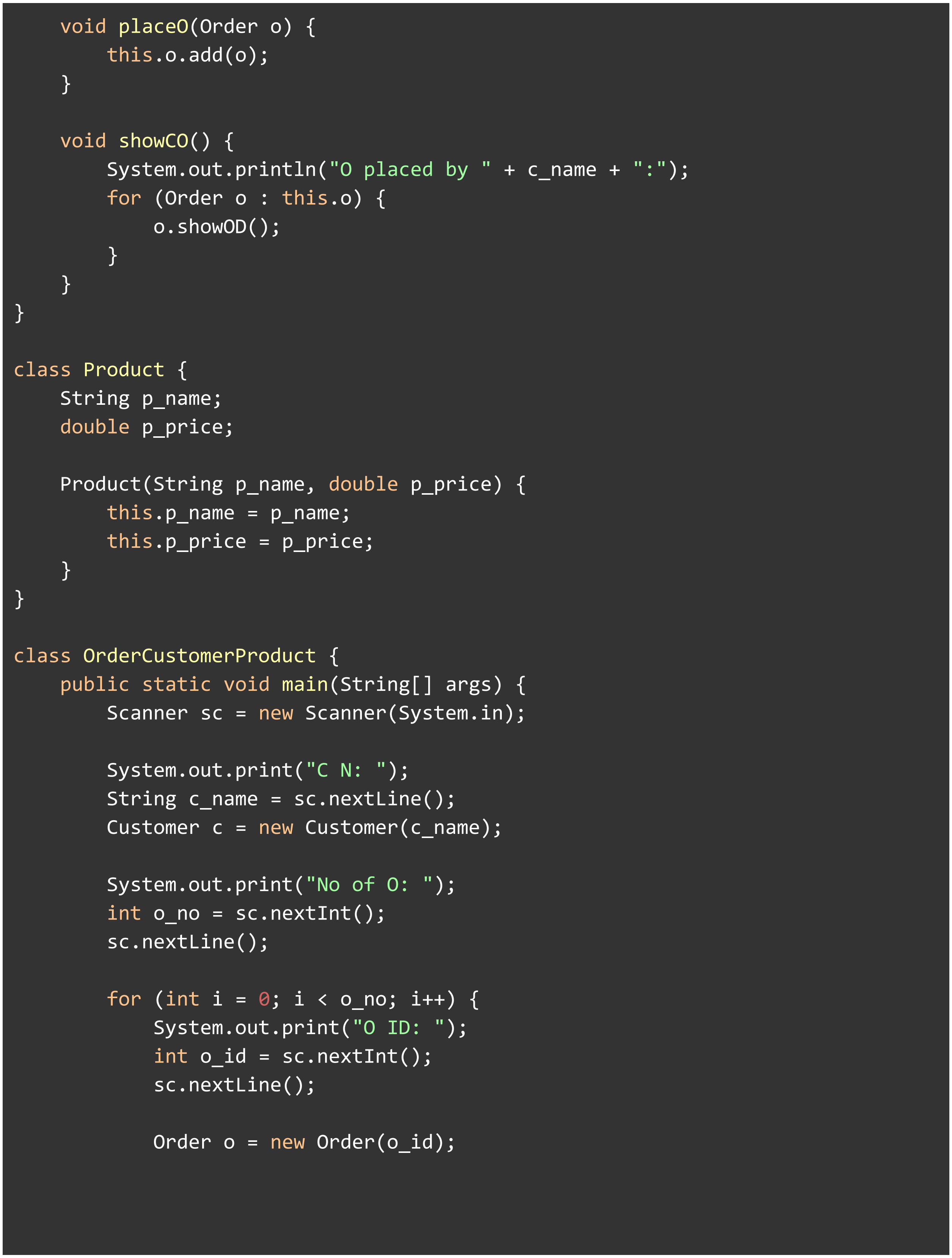


```
        for (int i = 0; i < doctorCount; i++) {

            String doctorName = sc.nextLine();
            Doctor doctor = new Doctor(doctorName);
            hospital.addDoctor(doctor);
        }

        System.out.print("Enter number of patients in " + hospitalName + ": ");
        int patientCount = sc.nextInt();
        sc.nextLine();

        for (int i = 0; i < patientCount; i++) {

            String patientName = sc.nextLine();
            Patient patient = new Patient(patientName);
            hospital.addPatient(patient);
        }

        hospital.showDoctors();

        hospital.showPatients();

        for (Doctor doctor : hospital.docs) {
            for (Patient patient : hospital.pts) {
                doctor.consult(patient);
            }
        }

        sc.close();
    }
}
```

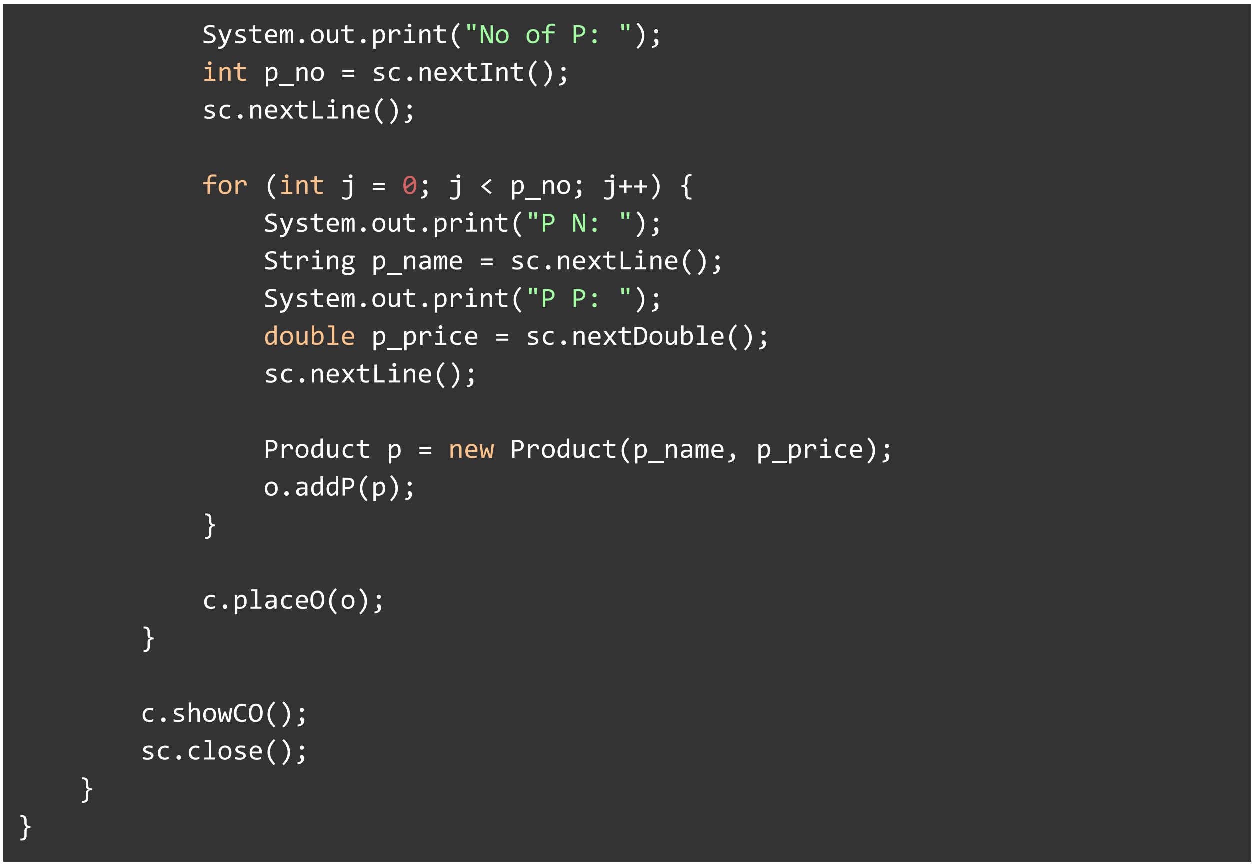
### Problem 4: E-commerce Platform with Orders, Customers, and Products

- **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.

```
import java.util.*;

class Order {
    int o_id;
    List<Product> p = new ArrayList<>();
    double t;

    Order(int o_id) {
        this.o_id = o_id;
    }

    void addP(Product p) {
        this.p.add(p);
        t += p.p_price;
    }

    void showOD() {
        System.out.println("O ID: " + o_id);
        for (Product p : this.p) {
            System.out.println("P: " + p.p_name + ", P: " + p.p_price);
        }
        System.out.println("T: " + t);
    }
}

class Customer {
    String c_name;
    List<Order> o = new ArrayList<>();

    Customer(String c_name) {
        this.c_name = c_name;
    }
```



```
    void placeO(Order o) {
        this.o.add(o);
    }

    void showCO() {
        System.out.println("O placed by " + c_name + ":");
        for (Order o : this.o) {
            o.showOD();
        }
    }
}

class Product {
    String p_name;
    double p_price;

    Product(String p_name, double p_price) {
        this.p_name = p_name;
        this.p_price = p_price;
    }
}

class OrderCustomerProduct {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("C N: ");
        String c_name = sc.nextLine();
        Customer c = new Customer(c_name);

        System.out.print("No of O: ");
        int o_no = sc.nextInt();
        sc.nextLine();

        for (int i = 0; i < o_no; i++) {
            System.out.print("O ID: ");
            int o_id = sc.nextInt();
            sc.nextLine();

            Order o = new Order(o_id);
```

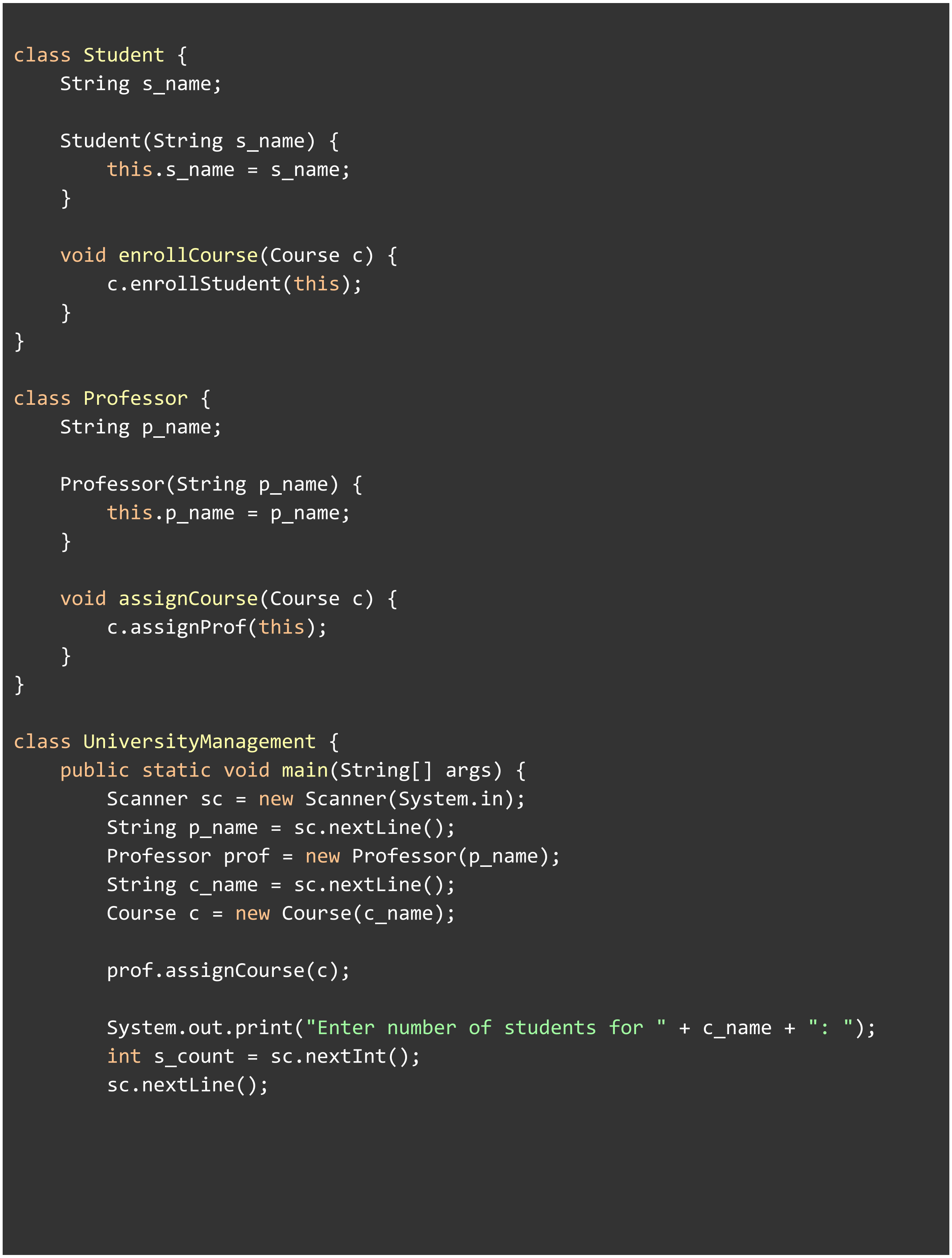


```
            System.out.print("No of P: ");
            int p_no = sc.nextInt();
            sc.nextLine();

            for (int j = 0; j < p_no; j++) {
                System.out.print("P N: ");
                String p_name = sc.nextLine();
                System.out.print("P P: ");
                double p_price = sc.nextDouble();
                sc.nextLine();

                Product p = new Product(p_name, p_price);
                o.addP(p);
            }

            c.placeO(o);
        }

        c.showCO();
        sc.close();
    }
}
```

**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.

```java
import java.util.*;

class Course {
    String c_name;
    Professor prof;
    List<Student> students = new ArrayList<>();

    Course(String c_name) {
        this.c_name = c_name;
    }

    void assignProf(Professor p) {
        prof = p;
    }

    void enrollStudent(Student s) {
        students.add(s);
    }

    void showCourseDetails() {
        System.out.println("Course: " + c_name);
        System.out.println("Professor: " + prof.p_name);
        System.out.println("Enrolled Students:");
        for (Student s : students) {
            System.out.println("- " + s.s_name);
        }
    }
}
```



```java
class Student {
    String s_name;

    Student(String s_name) {
        this.s_name = s_name;
    }

    void enrollCourse(Course c) {
        c.enrollStudent(this);
    }
}

class Professor {
    String p_name;

    Professor(String p_name) {
        this.p_name = p_name;
    }

    void assignCourse(Course c) {
        c.assignProf(this);
    }
}

class UniversityManagement {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String p_name = sc.nextLine();
        Professor prof = new Professor(p_name);
        String c_name = sc.nextLine();
        Course c = new Course(c_name);

        prof.assignCourse(c);

        System.out.print("Enter number of students for " + c_name + ": ");
        int s_count = sc.nextInt();
        sc.nextLine();
        for (int i = 0; i < s_count; i++) {
            String s_name = sc.nextLine();
            Student s = new Student(s_name);
            s.enrollCourse(c);
        }

        c.showCourseDetails();
        sc.close();
    }
}
```