**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 a List<Book> collection.
* 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.

| using System; using System.Collections.Generic;  class Book {   public string title;  public string author;   // Constructor  public Book(string title, string author)  {   this.title = title;  this.author = author;  }   // Method to display book details  public void DisplayBook()  {  Console.WriteLine("this.title: {0}, this.author: {1}" , this.title , this.author);  } }  // Library class class Library {  // List to hold books  private readonly List<Book> books;    public string libraryName;    // Constructor  public Library(string name)  {   libraryName = name;  books = new List<Book>();  }   // Method to add a book  public void AddBook(Book book)  {   books.Add(book);  }   // Method to display all books  public void DisplayLibraryBooks()  {  Console.WriteLine("Library: {0} contains the following books:" , libraryName);  if (books.Count == 0)  {  Console.WriteLine("No books available in the library.");  return;  }  foreach (var book in books)  {  book.DisplayBook();  }  } }  // Main class (aggregation) class Program {  static void Main()  {    // Creating Book objects  Book book1 = new Book("The Last Kingdom", "Rohit");  Book book2 = new Book("The Last Kingdom: Revolution", "Atul");  Book book3 = new Book("Falling Kingdom", "Aman");   // Creating Library objects  Library library1 = new Library("GLA Central Library");  Library library2 = new Library("GLA MBA Library");   // Adding books to different libraries  library1.AddBook(book1);  library1.AddBook(book2);    library2.AddBook(book2);  library2.AddBook(book3);   // Display books in each library  library1.DisplayLibraryBooks();  library2.DisplayLibraryBooks();    } } |
| --- |

**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.

| using System; using System.Collections.Generic;  // Bank class class Bank {  public string bankName;  private List<Customer> customers;   // Constructor  public Bank(string bankName)  {  this.bankName = bankName;  customers = new List<Customer>();  }   // Method to open an account  public void OpenAccount(Customer customer, decimal deposit)  {  Account newAccount = new Account(this, deposit);  customer.AddAccount(newAccount);  customers.Add(customer);  }   // Method to display customers  public void DisplayCustomers()  {  Console.WriteLine("Bank: {0} - Customer Accounts:" , bankName);  foreach (var customer in customers)  {  customer.ViewAccounts();  }  } }  class Customer {  public string name;  private List<Account> accounts;   // Constructor  public Customer(string name)  {  this.name = name;  accounts = new List<Account>();  }   // Method to add an account  public void AddAccount(Account account)  {  accounts.Add(account);  }   // Method to view all accounts  public void ViewAccounts()  {  Console.WriteLine("Customer: {0}" , Name);  foreach (var account in accounts)  {  account.DisplayAccountInfo();  }  } }  class Account {  private static int accountNumberSeed = 1000;  public int accountNumber;  public Bank bank;  public decimal balance;   // Constructor  public Account(Bank bank, decimal deposit)  {  this.bank = bank;  this.balance = deposit;  this.accountNumber = accountNumberSeed++;  }   // Method to display account details  public void DisplayAccountInfo()  {  Console.WriteLine("Account Number: {0}, Bank: {1}, balance: {2}" , accountNumber , Bank.bankName , balance);  } }  // Main class (association) class Program {  static void Main()  {    // Creating a bank  Bank bank1 = new Bank("INDIAN BANK");    // Creating customers  Customer customer1 = new Customer("Rohit Dixit");  Customer customer2 = new Customer("Rahul Dixit");    // Opening accounts for customers  bank1.OpenAccount(customer1, 1000);  bank1.OpenAccount(customer2, 1500);    // Display all customers  bank1.DisplayCustomers();   } } |
| --- |

**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.

| using System; using System.Collections.Generic;  // Employee class class Employee {  public string name;  public string position;   // Constructor  public Employee(string name, string position)  {  this.name = name;  this.position = position;  }   // Method to display details  public void DisplayEmployee()  {  Console.WriteLine("Employee: {0}, position: {1}", this.name , this.position);  } }  // Department class class Department {  public string departmentName;  private List<Employee> employees;   // Constructor  public Department(string departmentName)  {  this.departmentName = departmentName;  this.employees = new List<Employee>();  }   // Method to add an employee  public void AddEmployee(string name, string position)  {  employees.Add(new Employee(name, position));  }   // Method to display all employees  public void DisplayDepartment()  {  Console.WriteLine("Department: {0}", departmentName);  if (employees.Count == 0)  {  Console.WriteLine("No employees in this department.");  return;  }  foreach (var employee in employees)  {  employee.DisplayEmployee();  }  } }  // Company class class Company {  public string companyName;  private List<Department> departments;   // Constructor  public Company(string companyName)  {  this.companyName = companyName;  this.departments = new List<Department>();  }   // Method to add a department  public void AddDepartment(string departmentName)  {  departments.Add(new Department(departmentName));  }   // Method to get a department  public Department GetDepartment(string departmentName)  {  foreach (var dept in departments)  {  if (dept.departmentName == departmentName)  return dept;  }   return null;  }   // Method to display all information  public void DisplayCompany()  {  Console.WriteLine("Company: {0}" , companyName);  if (departments.Count == 0)  {  Console.WriteLine("No departments in the company.");  return;  }  foreach (var department in departments)  {  department.DisplayDepartment();  }  } }  // Main class (composition) class Program {  public static void Main()  {  // Creating company  Company company = new Company("X-park Technologies");    // Adding departments  company.AddDepartment("Technical");  company.AddDepartment("Human Resources");    // Adding employees to departments  Department technical = company.GetDepartment("Technical");  technical.AddEmployee("Vansh Saxena", "Software Analyst");  technical.AddEmployee("Krishana", "CTO");     Department hr = company.GetDepartment("Human Resources");  hr.AddEmployee("Rahul Kumar", "HR Manager");     // Displaying Company info  company.DisplayCompany();  } } |
| --- |

**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.

using System;

using System.Collections.Generic;

// Course class

class Course

{

public string courseName;

private List<Student> students;

// Constructor

public Course(string courseName)

{

this.courseName = courseName;

this.students = new List<Student>();

}

// Method to enroll a student

public void EnrollStudent(Student student)

{

students.Add(student);

}

// Method to display enrolled students

public void DisplayStudents()

{

Console.WriteLine("Course: {0}", courseName);

if (students.Count == 0)

{

Console.WriteLine("No students enrolled in this course.");

return;

}

foreach (var student in students)

{

Console.WriteLine("Student: {0}", student.studentName);

}

}

}

// Student class

class Student

{

public string studentName;

private List<Course> courses;

// Constructor

public Student(string studentName)

{

this.studentName = studentName;

this.courses = new List<Course>();

}

// Method to enroll in a course

public void EnrollInCourse(Course course)

{

courses.Add(course);

course.EnrollStudent(this);

}

// Method to display enrolled courses

public void DisplayCourses()

{

Console.WriteLine("Student: {0}", studentName);

if (courses.Count == 0)

{

Console.WriteLine("No courses enrolled.");

return;

}

foreach (var course in courses)

{

Console.WriteLine("Course: {0}", course.courseName);

}

}

}

// School class

class School

{

public string schoolName;

private List<Student> students;

// Constructor

public School(string schoolName)

{

this.schoolName = schoolName;

this.students = new List<Student>();

}

// Method to add a student

public void AddStudent(Student student)

{

students.Add(student);

}

// Method to display all students

public void DisplayStudents()

{

Console.WriteLine("School: {0}", schoolName);

if (students.Count == 0)

{

Console.WriteLine("No students in the school.");

return;

}

foreach (var student in students)

{

student.DisplayCourses();

}

}

}

// Main class

class Program

{

static void Main()

{

// Creating school

School school = new School("Kendriya Vidyalaya");

// Creating students

Student rohit = new Student("Rohti");

Student mohit = new Student("Mohit");

// Creating courses

Course math = new Course("Mathematics");

Course science = new Course("Science");

// Enrolling students in courses

rohit.EnrollInCourse(math);

rohit.EnrollInCourse(science);

mohit.EnrollInCourse(math);

// Adding students to the school

school.AddStudent(rohit);

school.AddStudent(mohit);

// Displaying school details

school.DisplayStudents();

// Displaying course details

math.DisplayStudents();

science.DisplayStudents();

}

}

**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.

| using System; using System.Collections.Generic;  // Faculty class class Faculty {  public string name;  public string designation;   // Constructor  public Faculty(string name, string designation)  {  this.name = name;  this.designation = designation;  }   // Display Faculty details  public void DisplayFaculty()  {  Console.WriteLine("Faculty: {0}, designation: {1}", this.name, this.designation);  } }  // Department class class Department {  public string departmentName ;  private List<Faculty> facultyMembers;   // Constructor  public Department(string departmentName)  {  this.departmentName = departmentName;  this.facultyMembers = new List<Faculty>();  }   // Method to add a Faculty  public void AddFaculty(Faculty faculty)  {  facultyMembers.Add(faculty);  }   // Display department  public void DisplayDepartment()  {  Console.WriteLine("Department: {0}", this.departmentName);  if (facultyMembers.Count == 0)  {  Console.WriteLine("No faculty members in this department");  return;  }  foreach (var faculty in facultyMembers)  {  faculty.DisplayFaculty();  }  } }   // University class  class University {  public string universityName;  private List<Department> departments;   // Constructor  public University(string universityName)  {  this.universityName = universityName;  this. departments = new List<Department>();  }   // Method to add a department  public void AddDepartment(string departmentName)  {  departments.Add(new Department(departmentName));  }   // Method to get a department  public Department GetDepartment(string departmentName)  {  foreach (var dept in departments)  {  if (dept.departmentName == departmentName)  return dept;  }  return null;  }   // Display university details  public void DisplayUniversity()  {  Console.WriteLine("University: {0}", this.universityName);  if (departments.Count == 0)  {  Console.WriteLine("No departments in the university");  return;  }  foreach (var department in departments)  {  department.DisplayDepartment();  }  } }  // Main class class Program {  static void Main()  {  // Creating university  University university = new University("GLA University");    // Adding departments  university.AddDepartment("Computer Science");  university.AddDepartment("Physics");    // Creating faculty members  Faculty prof1 = new Faculty("Rohit Dixit", "Professor");  Faculty prof2 = new Faculty("Mohit Singh", "Assistant Professor");    // Assigning faculty to departments  Department cs = university.GetDepartment("Computer Science");  cs.AddFaculty(prof1);    Department physics = university.GetDepartment("Physics");  physics.AddFaculty(prof2);    // Displaying info  university.DisplayUniversity();  } } |
| --- |

**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.

| using System; using System.Collections.Generic;  // Patient class class Patient {  public string name;  public int age;    public Patient(string name, int age)  {  this.name = name;  this.age = age;  }    public void DisplayPatient()  {  Console.WriteLine("Patient: {0}, age: {1}", this.name, this.age);  } }  // Doctor class class Doctor {  public string name;  public string specialty;  private List<Patient> patients;    // constructor  public Doctor(string name, string specialty)  {  this.name = name;  this.specialty = specialty;  this.patients = new List<Patient>();  }    // method to add Patient  public void AddPatient(Patient patient)  {  patients.Add(patient);  }    public void Consult(Patient patient)  {  Console.WriteLine("Doctor {0} ({1}) is consulting with Patient {2}", name, specialty, patient.name);  }    // Display Doctor Details  public void DisplayDoctor()  {  Console.WriteLine("Doctor: {0}, specialty: {1}", name, specialty);  Console.WriteLine("Patients:");  foreach (var patient in patients)  {  patient.DisplayPatient();  }  } }  // Hospital class class Hospital {  public string name;  private List<Doctor> doctors;  private List<Patient> patients;    // constructor  public Hospital(string name)  {  this.name = name;  this.doctors = new List<Doctor>();  this.patients = new List<Patient>();  }    // method to add Doctor  public void AddDoctor(Doctor doctor)  {  doctors.Add(doctor);  }    // method to add Patient  public void AddPatient(Patient patient)  {  patients.Add(patient);  }    // Display Hospital Details  public void DisplayHospital()  {  Console.WriteLine("Hospital: {0}", name);  Console.WriteLine("Doctors:");  foreach (var doctor in doctors)  {  doctor.DisplayDoctor();  }  Console.WriteLine("Patients:");  foreach (var patient in patients)  {  patient.DisplayPatient();  }  } }  // Main Class class Program {  public static void Main()  {  Hospital hospital = new Hospital("Maxx Hospital");    Doctor doctor1 = new Doctor("Dr. Abhay", "Radiologist");  Doctor doctor2 = new Doctor("Dr. Shiv kumar", "Dentist");    Patient patient = new Patient("Vansh", 25);    hospital.AddDoctor(doctor1);  hospital.AddDoctor(doctor2);  hospital.AddPatient(patient);    doctor1.AddPatient(patient);    doctor1.Consult(patient);    hospital.DisplayHospital();  } } |
| --- |

**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.

| using System; using System.Collections.Generic;  // Product class class Product {  public string productName;  public decimal price;   // constructor  public Product(string productName, decimal price)  {  this.productName = productName;  this.price = price;  }   // method to display product  public void DisplayProduct()  {  Console.WriteLine("Product: {0}, price: {1:C}", productName, price);  } }  // Order class class Order {  public int orderId;  private List<Product> products;   // constructor  public Order(int orderId)  {  this.orderId = orderId;  this.products = new List<Product>();  }   // method to Add Product  public void AddProduct(Product product)  {  products.Add(product);  }   // method to Display Orders  public void DisplayOrder()  {  Console.WriteLine("Order ID: {0}", orderId);  foreach (var product in products)  {  product.DisplayProduct();  }  } }  // Customer class placing orders class Customer {  public string customerName;  private List<Order> orders;   // customer  public Customer(string customerName)  {  this.customerName = customerName;  this. orders = new List<Order>();  }   // method to Place Order  public void PlaceOrder(Order order)  {  orders.Add(order);  }   // method to display All Customers Orders  public void DisplayCustomerOrders()  {  Console.WriteLine("Customer: {0}", this.customerName);  foreach (var order in orders)  {  order.DisplayOrder();  }  } }  // Main class class Program {  static void Main()  {  // Creating products  Product product1 = new Product("Shoes", 800);  Product product2 = new Product("Smart and Handsome Cream", 50);    // Creating an order object  Order order = new Order(101);  order.AddProduct(product1);  order.AddProduct(product2);    // Creating a customer objectu  Customer customer = new Customer("Vansh Saxena");  customer.PlaceOrder(order);    // Displaying customer orders  customer.DisplayCustomerOrders();  } } |
| --- |

**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.

| using System; using System.Collections.Generic;  // Student class class Student {  public string name;  private List<Course> enrolledCourses;   // Constructor  public Student(string name)  {  this. name = name;  this.enrolledCourses = new List<Course>();  }   // Method to enroll a student  public void EnrollCourse(Course course)  {  enrolledCourses.Add(course);  course.AddStudent(this);  }   // Method to display all courses  public void DisplayCourses()  {  Console.WriteLine("Student: {0} is enrolled in:", name);  foreach (var course in enrolledCourses)  {  Console.WriteLine("- {0}", course.courseName);  }  } }  // Professor class class Professor {  public string name;   // Constructor  public Professor(string name)  {  this.name = name;  } }  // Course class class Course {  public string courseName;  private List<Student> students;  public Professor assignedProfessor;   // Constructor  public Course(string courseName)  {  this.courseName = courseName;  this.students = new List<Student>();  }   // Method to assign a professor  public void AssignProfessor(Professor professor)  {  assignedProfessor = professor;  }   // Method to add a student  public void AddStudent(Student student)  {  students.Add(student);  }   // Method to display course details  public void DisplayCourseInfo()  {  Console.WriteLine("Course: {0}", this.courseName);  Console.WriteLine("Professor: {0}", assignedProfessor.name);  Console.WriteLine("Enrolled Students:");  foreach (var student in students)  {  Console.WriteLine("{0}", student.name);  }  } }  // Main Class class Program {  static void Main()  {  // Creating student object  Student student1 = new Student("Rohit");  Student student2 = new Student("Mohit");    // Creating professor object  Professor professor = new Professor("Dr. KashiNath");    // Creating course object  Course course = new Course("Data Structure");  course.AssignProfessor(professor);    // Enrolling students  student1.EnrollCourse(course);  student2.EnrollCourse(course);    // Displaying course information  course.DisplayCourseInfo();  } } |
| --- |