1. Library Management System

Project Description:
This project simulates a simple library system where a collection of books is managed. Students will create classes to represent individual books and the library. The system allows adding, removing, searching, and listing books.

Class: Book

- **Attributes:**
  - `title: String`
  - `author: String`
  - `isbn: String`
  - `available: boolean`
- **Methods:**
  - `getTitle() -> String`
    Returns the title of the book.
  - `isAvailable() -> boolean`
    Returns whether the book is currently available for borrowing.

Class: Library

- **Attributes:**
  - `books: List<Book>`
- **Methods:**
  - `addBook(book: Book) -> void`
    Adds a new book to the library's collection.
  - `removeBook(isbn: String) -> boolean`
    Removes a book identified by its ISBN from the collection. Returns true if removal is successful, false otherwise.
  - `searchBookByTitle(title: String) -> Book`
    Searches for a book by its title and returns the first matching book, or null/None if not found.
  - `listAllBooks() -> List<Book>`
    Returns the list of all books in the library.

2. Parking Lot Management

Project Description:

This project models a parking lot system. Students will manage a list of cars parked in a lot. The system should support parking a new car, removing a parked car, and displaying all parked cars.

Class: Car

- **Attributes:**
  - `licensePlate: String`
  - `model: String`
  - `parkingTime: String` *(or int for minutes)*
- **Methods:**
  - `getLicensePlate() -> String`
    Returns the car's license plate number.

Class: ParkingLot

- **Attributes:**
  - `parkedCars: List<Car>`
- **Methods:**
  - `parkCar(car: Car) -> void`
    Adds a car to the parking lot (i.e., to the parkedCars list).
  - `removeCar(licensePlate: String) -> boolean`
    Removes the car with the given license plate from the lot. Returns true if successful, otherwise false.
  - `displayCars() -> List<Car>`
    Returns the list of currently parked cars.

3. Student Course Enrollment

Project Description:
This project implements a student enrollment system. It manages students and courses, allowing students to enroll in courses while checking for available capacity. It also lists all students in a particular course.

Class: Student

- **Attributes:**
  - `studentID: int`
  - `name: String`
  - `enrolledCourses: List<String>` *(course codes)*
- **Methods:**

- `enroll(courseCode: String) -> void`
  Adds a course code to the student's list of enrolled courses.
- `getCourses() -> List<String>`
  Returns a list of courses in which the student is enrolled.

Class: Course

- **Attributes:**
  - `courseCode: String`
  - `courseName: String`
  - `maxEnrollment: int`
  - `currentEnrollment: int`
- **Methods:**
  - `canEnroll() -> boolean`
    Checks if the course has room for more students (i.e., if currentEnrollment is less than maxEnrollment) and returns true if enrollment is possible.

Class: EnrollmentManager

- **Attributes:**
  - `students: List<Student>`
  - `courses: List<Course>`
- **Methods:**
  - `enrollStudent(studentID: int, courseCode: String) -> boolean`
    Enrolls a student in a course if possible by updating the student's enrolledCourses and the course's currentEnrollment. Returns true if successful, false otherwise.
  - `listStudentsInCourse(courseCode: String) -> List<Student>`
    Returns a list of all students enrolled in the given course.

4. Bank Customer and Account Management

Project Description:
This project simulates basic banking operations. It allows management of bank accounts for customers. Students will implement methods to deposit, withdraw, and check account balances, as well as associate multiple accounts with a customer.

Class: BankAccount

- **Attributes:**
  - `accountNumber: String`

- `balance: double`
- `accountType: String` *(e.g., "savings", "checking")*

- **Methods:**
  - `deposit(amount: double) -> void`
    Increases the account balance by the given amount.
  - `withdraw(amount: double) -> boolean`
    Attempts to subtract the given amount from the balance. Returns true if successful (sufficient funds), otherwise false.
  - `getBalance() -> double`
    Returns the current balance of the account.

Class: Customer

- **Attributes:**
  - `customerID: int`
  - `name: String`
  - `accounts: List<BankAccount>`
- **Methods:**
  - `addAccount(account: BankAccount) -> void`
    Adds a bank account to the customer's list of accounts.
  - `getAccount(accountNumber: String) -> BankAccount`
    Searches for and returns the bank account with the specified account number, or null/None if not found.

5. Product Inventory Management

Project Description:
In this project, students will build an inventory management system to track products in a store. The system supports adding products, updating their stock levels, and listing all available products.

Class: Product

- **Attributes:**
  - `productID: int`
  - `name: String`
  - `price: double`
  - `quantityInStock: int`
- **Methods:**

- `updateStock(quantity: int) -> void`
  Updates the quantity in stock by adding the provided quantity (can be negative for deduction).
- `getProductInfo() -> String`
  Returns a string containing details about the product.

Class: Inventory

- **Attributes:**
  - `products: List<Product>`
- **Methods:**
  - `addProduct(product: Product) -> void`
    Adds a new product to the inventory list.
  - `updateProductStock(productID: int, quantity: int) -> boolean`
    Finds the product by productID and updates its stock by the specified quantity. Returns true if the product exists and was updated, otherwise false.
  - `listProducts() -> List<Product>`
    Returns the list of all products in the inventory.

6. Movie Ticket Booking System

Project Description:
This project models a movie ticket booking system. Students will create classes to manage movies, tickets, and bookings. The system should allow ticket booking, cancellation, and display of booking details.

Class: Movie

- **Attributes:**
  - `movieID: int`
  - `title: String`
  - `duration: int` *(in minutes)*
- **Methods:**
  - `getTitle() -> String`
    Returns the title of the movie.

Class: Ticket

- **Attributes:**
  - `ticketID: int`

- `movieID: int`
- `seatNumber: String`
- `price: double`

- **Methods:**
  - `getTicketInfo() -> String`
    Returns a string with details about the ticket (seat, movie, price, etc.).

Class: Booking

- **Attributes:**
  - `tickets: List<Ticket>`
- **Methods:**
  - `bookTicket(ticket: Ticket) -> void`
    Adds a new ticket to the booking list.
  - `cancelTicket(ticketID: int) -> boolean`
    Removes the ticket with the given ID from the booking list. Returns true if successful, false otherwise.
  - `getBookingDetails() -> List<Ticket>`
    Returns the list of all booked tickets.

7. Employee Payroll System

Project Description:
This project focuses on processing employee payrolls. Students will create classes for employees and a payroll processor to calculate and return the pay based on the hours worked and hourly rates.

Class: Employee

- **Attributes:**
  - `employeeID: int`
  - `name: String`
  - `hourlyRate: double`
- **Methods:**
  - `calculatePay(hoursWorked: double) -> double`
    Calculates and returns the pay for the employee based on the number of hours worked multiplied by the hourly rate.

Class: Payroll

- **Attributes:**
  - `employees: List<Employee>`
- **Methods:**
  - `processPayroll(hoursMap: Map<int, double>) -> Map<int, double>`
    Takes a mapping of employee IDs to hours worked, computes each employee's pay using the calculatePay method, and returns a new map with employee IDs and their corresponding calculated pay.

8. Flight Reservation System

Project Description:
This project simulates a flight reservation system where flights and reservations are managed. Students will implement methods to reserve a seat on a flight, cancel reservations, and view reservation details.

Class: Flight

- **Attributes:**
  - `flightNumber: String`
  - `origin: String`
  - `destination: String`
  - `seatsAvailable: int`
- **Methods:**
  - `reserveSeat() -> boolean`
    Checks if there is an available seat. If so, decrements seatsAvailable and returns true; otherwise, returns false.

Class: Reservation

- **Attributes:**
  - `reservationID: int`
  - `flightNumber: String`
  - `passengerName: String`
- **Methods:**
  - `getReservationDetails() -> String`
    Returns a string containing details of the reservation.

Class: ReservationManager

- **Attributes:**

- `flights: List<Flight>`
- `reservations: List<Reservation>`

- **Methods:**
  - `makeReservation(flightNumber: String, passengerName: String) -> Reservation`
    Creates a reservation for the specified flight and passenger. Returns the Reservation object if the seat is successfully reserved.
  - `cancelReservation(reservationID: int) -> boolean`
    Cancels the reservation with the given ID. Returns true if cancellation is successful, false otherwise.

9. Restaurant Order Management

Project Description:
This project models an order management system for a restaurant. The system involves managing a menu of items, processing orders, and calculating the total bill for an order.

Class: MenuItem

- **Attributes:**
  - `itemID: int`
  - `name: String`
  - `price: double`
- **Methods:**
  - `getItemDetails() -> String`
    Returns a string describing the menu item, including its name and price.

Class: Order

- **Attributes:**
  - `orderID: int`
  - `items: List<MenuItem>`
  - `tableNumber: int`
- **Methods:**
  - `addItem(item: MenuItem) -> void`
    Adds a menu item to the order.
  - `calculateTotal() -> double`
    Calculates and returns the total price of all the items in the order.

Class: OrderManager

- **Attributes:**
  - `orders: List<Order>`
- **Methods:**
  - `createOrder(order: Order) -> void`
    Adds a new order to the list of orders.
  - `cancelOrder(orderID: int) -> boolean`
    Cancels an existing order by orderID. Returns true if the order was found and cancelled, otherwise false.
  - `getOrder(orderID: int) -> Order`
    Returns the order that matches the given orderID, or null/None if it does not exist.

10. Hospital Patient Management

Project Description:
This project manages patients and their appointments in a hospital setting. Students will create classes for patients and appointments, allowing scheduling, cancellation, and listing of appointments for a given patient.

Class: Patient

- **Attributes:**
  - `patientID: int`
  - `name: String`
  - `age: int`
- **Methods:**
  - `getPatientInfo() -> String`
    Returns a string with the patient's details (ID, name, and age).

Class: Appointment

- **Attributes:**
  - `appointmentID: int`
  - `patientID: int`
  - `doctorName: String`
  - `appointmentTime: String`
- **Methods:**
  - `getAppointmentDetails() -> String`
    Returns a string describing the appointment details.

Class: PatientManager

- **Attributes:**
  - `patients: List<Patient>`
  - `appointments: List<Appointment>`
- **Methods:**
  - `scheduleAppointment(appointment: Appointment) -> void`
    Adds a new appointment to the appointments list for a patient.
  - `cancelAppointment(appointmentID: int) -> boolean`
    Cancels an appointment by appointmentID. Returns true if the cancellation was successful, otherwise false.
  - `listAppointmentsForPatient(patientID: int) -> List<Appointment>`
    Returns all appointments associated with the specified patient ID.

11. Gym Membership System

Project Description:
This project simulates a gym membership management system. Students will create classes to register gym members, manage membership plans, and assign plans to members.

Class: Member

- **Attributes:**
  - `memberID: int`
  - `name: String`
  - `membershipType: String` *(e.g., "monthly", "yearly")*
- **Methods:**
  - `getMemberInfo() -> String`
    Returns a string with details about the gym member.

Class: MembershipPlan

- **Attributes:**
  - `planID: int`
  - `planName: String`
  - `fee: double`
- **Methods:**
  - `getPlanDetails() -> String`
    Returns a string containing the details of the membership plan (name and fee).

Class: Gym

- **Attributes:**
  - `members: List<Member>`
  - `plans: List<MembershipPlan>`
- **Methods:**
  - `registerMember(member: Member) -> void`
    Adds a new member to the gym's members list.
  - `assignPlan(memberID: int, planID: int) -> boolean`
    Assigns a membership plan to the member identified by memberID. Returns true if the assignment is successful, otherwise false.

12. University Admission System

Project Description:
This project simulates a university admission process. It involves managing applicants and their applications to various programs. The system should allow submission and review of applications.

Class: Applicant

- **Attributes:**
  - `applicantID: int`
  - `name: String`
  - `score: double`
- **Methods:**
  - `getApplicantInfo() -> String`
    Returns a string containing the applicant's details (ID, name, and score).

Class: Application

- **Attributes:**
  - `applicationID: int`
  - `applicantID: int`
  - `programApplied: String`
- **Methods:**
  - `getApplicationDetails() -> String`
    Returns a string with the details of the application, including the program applied for.

Class: AdmissionOffice

- **Attributes:**
  - `applicants: List<Applicant>`
  - `applications: List<Application>`
- **Methods:**
  - `submitApplication(application: Application) -> void`
    Adds a new application to the system.
  - `reviewApplication(applicantID: int) -> String`
    Reviews the application for the given applicantID and returns a decision or summary of the review.

13. Event Management System

Project Description:
This project simulates an event management system where events are organized, and participants are registered. The system should check if an event is full and list registered participants.

Class: Event

- **Attributes:**
  - `eventID: int`
  - `eventName: String`
  - `location: String`
  - `capacity: int`
- **Methods:**
  - `isEventFull() -> boolean`
    Determines whether the event has reached its capacity. Returns true if full, false otherwise.

Class: Participant

- **Attributes:**
  - `participantID: int`
  - `name: String`
  - `email: String`
- **Methods:**
  - `getParticipantInfo() -> String`
    Returns a string containing the participant's details.

Class: EventManager

- **Attributes:**
  - `events: List<Event>`
  - `participants: List<Participant>`
- **Methods:**
  - `registerParticipant(eventID: int, participant: Participant) -> boolean`
    Registers the participant for the event identified by eventID if the event is not full. Returns true if registration is successful, otherwise false.
  - `listParticipants(eventID: int) -> List<Participant>`
    Returns a list of participants registered for the given event.

14. Vehicle Rental System

Project Description:
This project creates a system for renting vehicles. Students will implement methods to rent and return vehicles, as well as record rental details for customers.

Class: Vehicle

- **Attributes:**
  - `vehicleID: int`
  - `model: String`
  - `rentalRate: double`
  - `isRented: boolean`
- **Methods:**
  - `rentVehicle() -> boolean`
    Checks if the vehicle is available for rent. If yes, marks it as rented and returns true; otherwise, returns false.
  - `returnVehicle() -> void`
    Marks the vehicle as available (i.e., not rented).

Class: Rental

- **Attributes:**
  - `rentalID: int`
  - `vehicleID: int`
  - `customerName: String`
  - `rentalDuration: int` *(in days)*
- **Methods:**

- `getRentalDetails() -> String`
  Returns a string with details about the rental (vehicle, customer, duration, etc.).

Class: RentalService

- **Attributes:**
  - `vehicles: List<Vehicle>`
  - `rentals: List<Rental>`
- **Methods:**
  - `createRental(rental: Rental) -> boolean`
    Creates a new rental if the vehicle is available. Returns true if the rental is successfully created, otherwise false.
  - `endRental(rentalID: int) -> boolean`
    Ends a rental by the given rentalID and marks the corresponding vehicle as available. Returns true if successful, otherwise false.

15. E-commerce Review System

Project Description:
This project models a review system for an e-commerce platform. It allows users to add reviews for products and retrieve reviews for a specific product.

Class: Review

- **Attributes:**
  - `reviewID: int`
  - `productID: int`
  - `reviewText: String`
  - `rating: int` *(e.g., 1-5)*
- **Methods:**
  - `getReviewSummary() -> String`
    Returns a short summary of the review, including the rating and a snippet of the text.

Class: Product

- **Attributes:**
  - `productID: int`
  - `name: String`
  - `price: double`

- **Methods:**
  - `getProductInfo() -> String`
    Returns a string with the product details, such as name and price.

Class: ReviewManager

- **Attributes:**
  - `reviews: List<Review>`
- **Methods:**
  - `addReview(review: Review) -> void`
    Adds a new review to the review list.
  - `getReviewsByProduct(productID: int) -> List<Review>`
    Returns a list of reviews associated with the specified product.

16. Banking Transaction Log

Project Description:
This project creates a system to log banking transactions. Students will implement a system that processes transactions (such as deposits or withdrawals) on bank accounts and maintains a transaction history.

Class: Transaction

- **Attributes:**
  - `transactionID: int`
  - `accountNumber: String`
  - `amount: double`
  - `transactionType: String` *(e.g., "deposit", "withdrawal")*
- **Methods:**
  - `getTransactionDetails() -> String`
    Returns a string summarizing the transaction details.

Class: BankAccount

- **Attributes:**
  - `accountNumber: String`
  - `balance: double`
  - `transactions: List<Transaction>`
- **Methods:**

- `performTransaction(transaction: Transaction) -> boolean`
  Processes a transaction (e.g., deposit or withdrawal), updates the balance accordingly, and adds the transaction to the transactions list. Returns true if the transaction was processed successfully, otherwise false.
- `getBalance() -> double`
  Returns the current account balance.
- `listTransactions() -> List<Transaction>`
  Returns the list of all transactions performed on the account.