**Курс:** Шаблоны проектирования приложений

**Тема:** Модуль 11 Диаграммы вариантов использования. Диаграмма классов

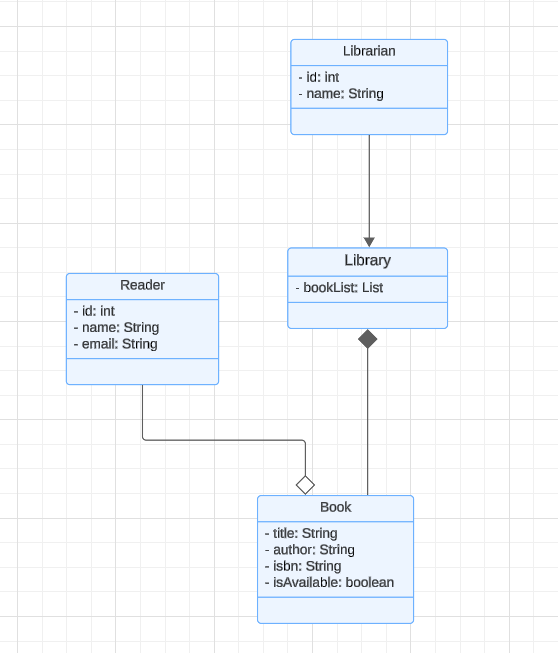
**Цель:**

Изучить и применить концепцию **диаграмм классов (Class Diagram)** для проектирования объектно-ориентированных систем. В рамках задания вы создадите диаграмму классов для простой системы управления библиотекой и реализуете её на языке C#.

**Описание задания:**

Разработайте систему управления библиотекой, которая включает такие сущности, как **Книга**, **Читатель**, и **Библиотекарь**. Система должна позволять читателям брать книги в аренду, возвращать их и предоставлять библиотекарю возможность управлять книгами.

**Часть 1: Создание диаграммы классов**

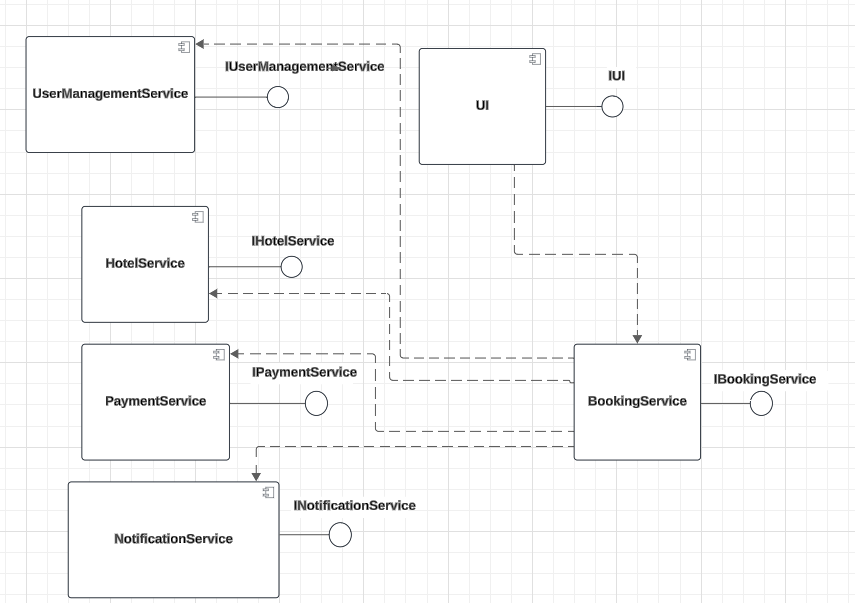
.

**Цель:** Научиться проектировать и визуализировать архитектуру распределенной системы, **используя диаграмму компонентов в UML**, а также реализовать ее основные элементы на языке C#.

**Описание задачи**

Разработайте архитектуру и реализацию распределенной системы бронирования отелей. В системе будет несколько компонентов, каждый из которых представляет собой отдельный модуль с определенной логикой. Необходимо спроектировать систему, в которой взаимодействие между компонентами происходит через интерфейсы, что позволит системе быть модульной и легко расширяемой.

**1.Диаграмма компонентов** с отображением всех компонентов, интерфейсов и связей



1. **Исходный код компонентов** с реализацией интерфейсов и их взаимодействием.

interface IUI {

void searchHotels(String location, String roomClass, double maxPrice);

void bookRoom(int userId, int hotelId, String startDate, String endDate);

void viewBookings(int userId);

}

interface IHotelService {

List<Hotel> searchHotels(String location, String roomClass, double maxPrice);

}

interface IBookingService {

Booking bookRoom(int userId, int hotelId, String startDate, String endDate);

boolean checkAvailability(int hotelId, String startDate, String endDate);

}

interface IPaymentService {

boolean processPayment(int bookingId, double amount, String paymentMethod);

}

interface INotificationService {

void sendNotification(int userId, String message);

}

interface IUserManagementService {

User register(String username, String password);

User login(String username, String password);

}

public class UI implements IUI {

private final IUserManagementService userManagementService;

private final IHotelService hotelService;

private final IBookingService bookingService;

private final IPaymentService paymentService;

private final INotificationService notificationService;

public UI(IUserManagementService userManagementService, IHotelService hotelService,

IBookingService bookingService, IPaymentService paymentService,

INotificationService notificationService) {

this.userManagementService = userManagementService;

this.hotelService = hotelService;

this.bookingService = bookingService;

this.paymentService = paymentService;

this.notificationService = notificationService;

}

@Override

public void searchHotels(String location, String roomClass, double maxPrice) {

List<Hotel> hotels = hotelService.searchHotels(location, roomClass, maxPrice);

hotels.forEach(System.out::println);

}

@Override

public void bookRoom(int userId, int hotelId, String startDate, String endDate) {

Booking booking = bookingService.bookRoom(userId, hotelId, startDate, endDate);

if (booking != null) {

notificationService.sendNotification(userId, "Your booking is confirmed.");

} else {

System.out.println("Room is not available.");

}

}

@Override

public void viewBookings(int userId) {

System.out.println("User bookings for userId " + userId);

}

}

public class HotelService implements IHotelService {

@Override

public List<Hotel> searchHotels(String location, String roomClass, double maxPrice) {

return List.of(new Hotel("Hotel1", location, roomClass, maxPrice));

}

}

public class BookingService implements IBookingService {

@Override

public Booking bookRoom(int userId, int hotelId, String startDate, String endDate) {

if (checkAvailability(hotelId, startDate, endDate)) {

return new Booking(userId, hotelId, startDate, endDate);

}

return null;

}

@Override

public boolean checkAvailability(int hotelId, String startDate, String endDate) {

return true;

}

}

public class PaymentService implements IPaymentService {

@Override

public boolean processPayment(int bookingId, double amount, String paymentMethod) {

System.out.println("Processing payment of " + amount + " for booking " + bookingId);

return true;

}

}

public class NotificationService implements INotificationService {

@Override

public void sendNotification(int userId, String message) {

System.out.println("Notification to user " + userId + ": " + message);

}

}

public class UserManagementService implements IUserManagementService {

@Override

public User register(String username, String password) {

return new User(username, password);

}

@Override

public User login(String username, String password) {

return new User(username, password);

}

}

public class Main {

public static void main(String[] args) {

IUserManagementService userManagementService = new UserManagementService();

IHotelService hotelService = new HotelService();

IBookingService bookingService = new BookingService();

IPaymentService paymentService = new PaymentService();

INotificationService notificationService = new NotificationService();

IUI ui = new UI(userManagementService, hotelService, bookingService, paymentService, notificationService);

ui.searchHotels("New York", "Deluxe", 300.0);

ui.bookRoom(1, 101, "2024-12-01", "2024-12-05");

}

}