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

1. Классы и их атрибуты:

Book (Книга):

Атрибуты: название, автор, ISBN, статус (в наличии/арендована).

Reader (Читатель):

Атрибуты: имя, список арендованных книг.

Librarian (Библиотекарь):

Атрибуты: имя.

Library (Библиотека):

Атрибуты: список книг.

2. Отношения между классами:

Класс Library имеет композицию с классом Book (библиотека содержит книги).

Класс Reader может арендовать несколько книг, следовательно, имеет связь с классом Book (агрегация).

Класс Librarian управляет книгами в библиотеке (связь с классом Library).

Ожидаемый результат:

Программа должна вывести информацию о доступных книгах в библиотеке, аренде книги читателем, а также об изменениях в списке доступных книг после аренды и возврата.

Дополнительные задания:

1. Добавьте возможность поиска книг по названию или автору.

2. Реализуйте функцию для отображения всех книг в библиотеке, включая те, которые находятся в аренде.

3. Добавьте ограничение на максимальное количество книг, которые читатель может взять в аренду одновременно.

import java.util.ArrayList;

import java.util.List;

class Book {

private String title;

private String author;

private String isbn;

private boolean isAvailable;

public Book(String title, String author, String isbn) {

this.title = title;

this.author = author;

this.isbn = isbn;

this.isAvailable = true;

}

public String getTitle() {

return title;

}

public String getAuthor() {

return author;

}

public boolean isAvailable() {

return isAvailable;

}

public void setAvailable(boolean available) {

isAvailable = available;

}

@Override

public String toString() {

return title + " by " + author + " (ISBN: " + isbn + ") - " + (isAvailable ? "Available" : "Rented");

}

}

class Reader {

private String name;

private List<Book> rentedBooks;

public Reader(String name) {

this.name = name;

this.rentedBooks = new ArrayList<>();

}

public boolean rentBook(Book book) {

if (rentedBooks.size() >= 3) {

System.out.println(name + " cannot rent more than 3 books.");

return false;

}

if (book.isAvailable()) {

book.setAvailable(false);

rentedBooks.add(book);

System.out.println(name + " rented \"" + book.getTitle() + "\".");

return true;

} else {

System.out.println("\"" + book.getTitle() + "\" is not available for rent.");

return false;

}

}

public void returnBook(Book book) {

if (rentedBooks.remove(book)) {

book.setAvailable(true);

System.out.println(name + " returned \"" + book.getTitle() + "\".");

} else {

System.out.println(name + " does not have \"" + book.getTitle() + "\".");

}

}

public void showRentedBooks() {

System.out.println(name + "'s rented books:");

for (Book book : rentedBooks) {

System.out.println("- " + book);

}

}

}

class Librarian {

private String name;

public Librarian(String name) {

this.name = name;

}

public void addBook(Library library, Book book) {

library.getBooks().add(book);

System.out.println("Librarian " + name + " added \"" + book.getTitle() + "\" to the library.");

}

}

class Library {

private List<Book> books;

public Library() {

books = new ArrayList<>();

}

public List<Book> getBooks() {

return books;

}

public void displayBooks() {

System.out.println("Books in the library:");

for (Book book : books) {

System.out.println("- " + book);

}

}

public Book searchBookByTitle(String title) {

for (Book book : books) {

if (book.getTitle().equalsIgnoreCase(title)) {

return book;

}

}

return null;

}

public List<Book> searchBooksByAuthor(String author) {

List<Book> result = new ArrayList<>();

for (Book book : books) {

if (book.getAuthor().equalsIgnoreCase(author)) {

result.add(book);

}

}

return result;

}

}

public class LibraryManagementSystem {

public static void main(String[] args) {

Library library = new Library();

Librarian librarian = new Librarian("Alice");

Reader reader = new Reader("John");

librarian.addBook(library, new Book("1984", "George Orwell", "1234567890"));

librarian.addBook(library, new Book("Brave New World", "Aldous Huxley", "0987654321"));

librarian.addBook(library, new Book("Fahrenheit 451", "Ray Bradbury", "1122334455"));

library.displayBooks();

Book searchedBook = library.searchBookByTitle("1984");

if (searchedBook != null) {

System.out.println("\nFound book: " + searchedBook);

}

reader.rentBook(searchedBook);

reader.rentBook(searchedBook);

reader.returnBook(searchedBook);

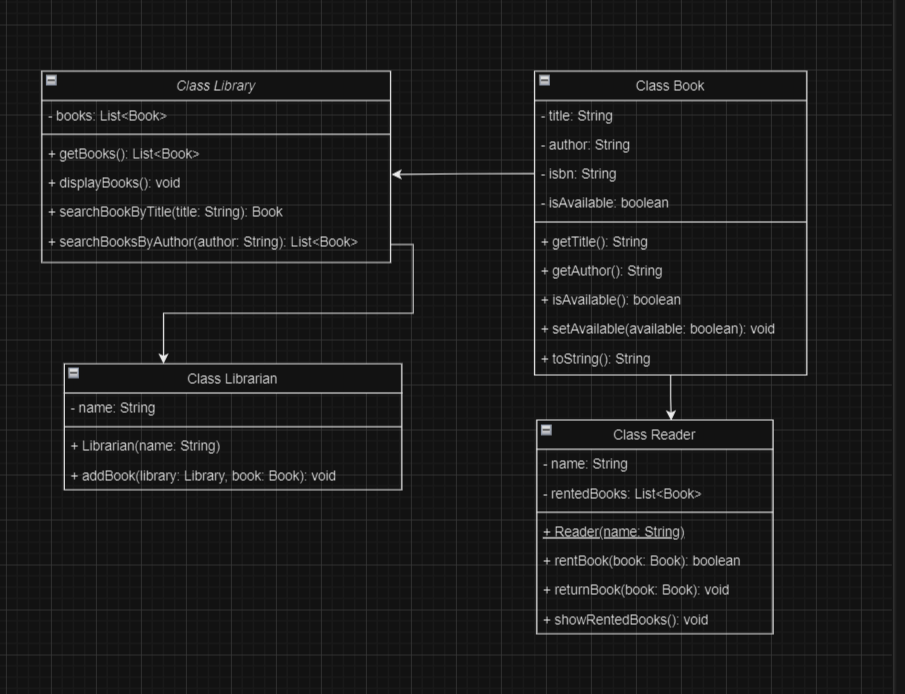
System.out.println("\nBooks after rental:");

library.displayBooks();

reader.showRentedBooks();

}

}



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.

Описание компонентов системы

1. UI-компонент – интерфейс для пользователей (консольное приложение), где пользователи могут искать отели, бронировать номера и просматривать свои бронирования.

2. HotelService – компонент для работы с данными отелей:

o Управляет информацией об отелях (название, местоположение, классы номеров, цены).

o Предоставляет методы для поиска отелей по заданным критериям (местоположение, класс, цена и т.д.).

3. BookingService – компонент для бронирования номеров:

o Осуществляет бронирование номера на определенные даты.

o Проверяет доступность номера.

o Хранит данные о бронированиях (пользователь, отель, даты).

4. PaymentService – компонент для обработки платежей:

o Управляет информацией о платежах.

o Проверяет оплату бронирований.

o Поддерживает оплату различными методами (например, карта, PayPal и т.д.).

5. NotificationService – компонент для отправки уведомлений пользователям:

o Отправляет уведомления о подтверждении бронирования.

o Отправляет напоминания перед заездом и подтверждения об оплате.

6. UserManagementService – компонент для управления данными пользователей:

o Регистрация и авторизация пользователей.

o Хранение информации о пользователях.

o Поддерживает методы для регистрации и входа.

Результаты выполнения:

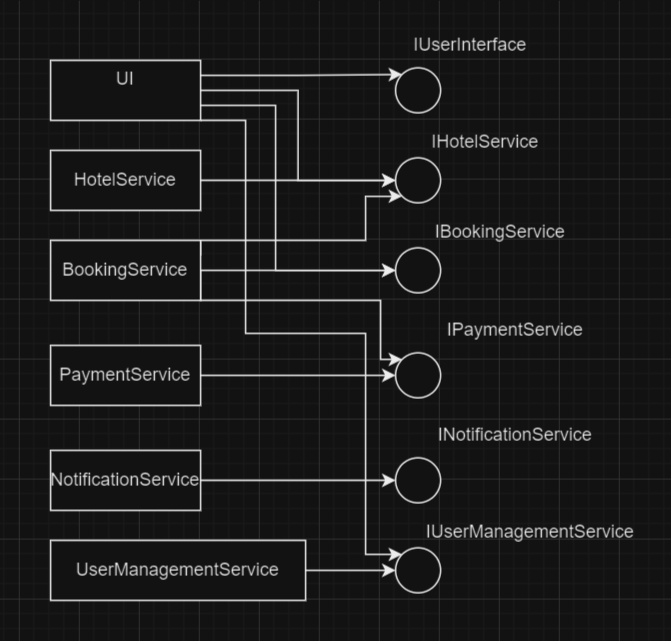
1. Диаграмма компонентов с отображением всех компонентов, интерфейсов и связей.

2. Исходный код компонентов с реализацией интерфейсов и их взаимодействием.

3. Консольный интерфейс, позволяющий пользователю взаимодействовать с системой.

4. Отчет, содержащий описание архитектуры системы, диаграмму компонентов, код и описание работы каждого компонента, а также результаты тестирования.

Итог: В результате выполнения данной работы вы изучите, как проектировать архитектуру приложения с помощью диаграммы компонентов, реализовывать взаимодействие компонентов через интерфейсы и создавать независимые модули на языке



interface IUserInterface {

void searchHotels(String location, String roomType, double price);

void bookRoom(String hotelId, String roomId, String startDate, String endDate);

void registerUser(String username, String password);

void loginUser(String username, String password);

}

interface IHotelService {

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

void addHotel(String hotelId, String location, String roomType, double price);

}

interface IBookingService {

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

}

interface IPaymentService {

boolean processPayment(String userId, String hotelId, String roomId);

}

interface INotificationService {

void sendBookingConfirmation(String userId, String hotelId, String roomId);

void sendReminder(String userId, String hotelId, String roomId);

void sendPaymentConfirmation(String userId, String hotelId, String roomId);

}

interface IUserManagementService {

void registerUser(String username, String password);

boolean authenticateUser(String username, String password);

}

class UIComponent implements IUserInterface {

private IHotelService hotelService;

private IBookingService bookingService;

private IUserManagementService userManagementService;

public UIComponent(IHotelService hotelService, IBookingService bookingService, IUserManagementService userManagementService) {

this.hotelService = hotelService;

this.bookingService = bookingService;

this.userManagementService = userManagementService;

}

public void searchHotels(String location, String roomType, double price) {

System.out.println("Searching for hotels...");

}

public void bookRoom(String hotelId, String roomId, String startDate, String endDate) {

bookingService.bookRoom("userId", hotelId, roomId, startDate, endDate);

}

public void registerUser(String username, String password) {

userManagementService.registerUser(username, password);

}

public void loginUser(String username, String password) {

userManagementService.authenticateUser(username, password);

}

}

class HotelService implements IHotelService {

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

return true;

}

public void addHotel(String hotelId, String location, String roomType, double price) {

System.out.println("Adding hotel: " + hotelId);

}

}

class BookingService implements IBookingService {

private IHotelService hotelService;

private IPaymentService paymentService;

public BookingService(IHotelService hotelService, IPaymentService paymentService) {

this.hotelService = hotelService;

this.paymentService = paymentService;

}

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

if (hotelService.checkAvailability(hotelId, roomId, startDate, endDate)) {

if (paymentService.processPayment(userId, hotelId, roomId)) {

System.out.println("Booking confirmed for user: " + userId);

}

} else {

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

}

}

}

class PaymentService implements IPaymentService {

public boolean processPayment(String userId, String hotelId, String roomId) {

System.out.println("Processing payment...");

return true;

}

}

class NotificationService implements INotificationService {

public void sendBookingConfirmation(String userId, String hotelId, String roomId) {

System.out.println("Sending booking confirmation...");

}

public void sendReminder(String userId, String hotelId, String roomId) {

System.out.println("Sending reminder...");

}

public void sendPaymentConfirmation(String userId, String hotelId, String roomId) {

System.out.println("Sending payment confirmation...");

}

}

class UserManagementService implements IUserManagementService {

public void registerUser(String username, String password) {

System.out.println("User registered: " + username);

}

public boolean authenticateUser(String username, String password) {

return true;

}

}

public class HotelBookingSystem {

public static void main(String[] args) {

IHotelService hotelService = new HotelService();

IPaymentService paymentService = new PaymentService();

IBookingService bookingService = new BookingService(hotelService, paymentService);

IUserManagementService userManagementService = new UserManagementService();

INotificationService notificationService = new NotificationService();

IUserInterface uiComponent = new UIComponent(hotelService, bookingService, userManagementService);

uiComponent.registerUser("user1", "password1");

uiComponent.loginUser("user1", "password1");

uiComponent.searchHotels("New York", "Single", 100);

uiComponent.bookRoom("hotel1", "room1", "2024-12-01", "2024-12-10");

}

}