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

**Лабораторная работа: Проектирование и реализация системы управления библиотекой с использованием диаграммы классов.**

**Цель работы:**

Изучить принципы построения диаграммы классов и применить их для разработки системы управления библиотекой.

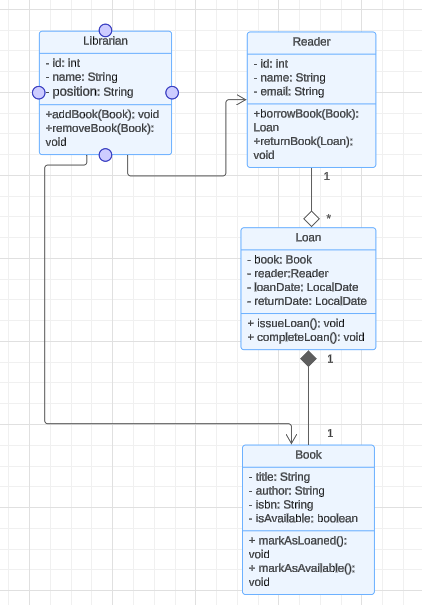
**Описание задания:**

Необходимо разработать систему управления библиотекой, которая включает следующие основные сущности:

1. **Книга** (Book) — хранит информацию о книгах.
2. **Читатель** (Reader) — представляет пользователей, которые могут брать книги.
3. **Библиотекарь** (Librarian) — управляет книгами и читателями.
4. **Выдача книги** (Loan) — регистрирует процесс выдачи книги.

**Этапы выполнения работы:**

**1. Проектирование диаграммы классов:**



**2. Реализация диаграммы:**

public 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 void markAsLoaned() {

isAvailable = false;

}

public void markAsAvailable() {

isAvailable = true;

}

public boolean isAvailable() {

return isAvailable;

}

}

import java.util.ArrayList;

import java.util.List;

public class Reader {

private int id;

private String name;

private String email;

private List<Loan> loans;

public Reader(int id, String name, String email) {

this.id = id;

this.name = name;

this.email = email;

this.loans = new ArrayList<>();

}

public Loan borrowBook(Book book) {

if (!book.isAvailable()) {

throw new IllegalStateException("Book is not available.");

}

Loan loan = new Loan(this, book);

loans.add(loan);

book.markAsLoaned();

return loan;

}

public void returnBook(Loan loan) {

loans.remove(loan);

loan.getBook().markAsAvailable();

loan.completeLoan();

}

}

import java.util.ArrayList;

import java.util.List;

public class Librarian {

private int id;

private String name;

private String position;

private List<Book> books;

public Librarian(int id, String name, String position) {

this.id = id;

this.name = name;

this.position = position;

this.books = new ArrayList<>();

}

public void addBook(Book book) {

books.add(book);

}

public void removeBook(Book book) {

books.remove(book);

}

public List<Book> getBooks() {

return books;

}

public void viewLoans() {

for (Book book : books) {

System.out.println(book.getTitle() + " - " + (book.isAvailable() ? "Available" : "Loaned"));

}

}

}

import java.time.LocalDate;

public class Loan {

private Book book;

private Reader reader;

private LocalDate loanDate;

private LocalDate returnDate;

public Loan(Reader reader, Book book) {

this.reader = reader;

this.book = book;

this.loanDate = LocalDate.now();

}

public void completeLoan() {

this.returnDate = LocalDate.now();

}

public Book getBook() {

return book;

}

public Reader getReader() {

return reader;

}

}

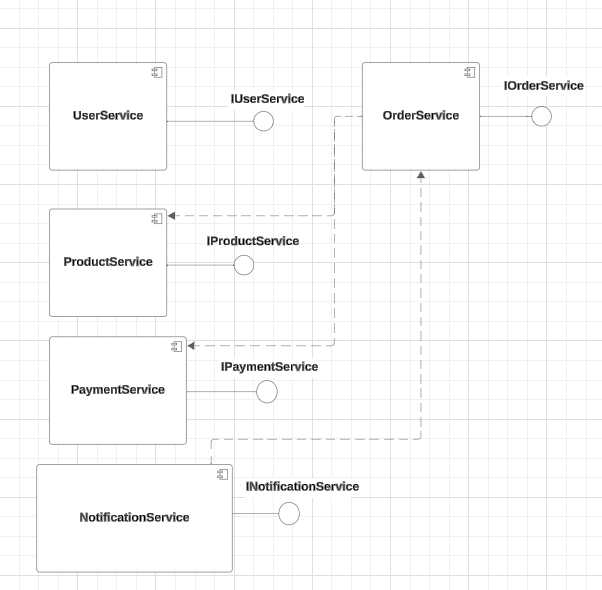
**Цель:**  
Научиться проектировать архитектуру приложения с использованием диаграммы компонентов UML и реализовать его основные элементы.

**Описание компонентов системы**

1. **UserService**: Компонент для управления пользователями.
   * Функции: регистрация пользователей, авторизация, управление информацией о пользователях.
2. **ProductService**: Компонент для управления товарами.
   * Функции: добавление и обновление товаров, получение списка товаров, поиск товаров по категории или названию.
3. **OrderService**: Компонент для управления заказами.
   * Функции: создание и управление заказами, отслеживание статуса заказа, отмена заказов.
4. **PaymentService**: Компонент для обработки платежей.
   * Функции: обработка платежей, проверка статуса платежа, уведомление об успешной оплате.
5. **NotificationService**: Компонент для отправки уведомлений.
   * Функции: отправка уведомлений пользователям о статусе заказов и платежей.

**Этапы выполнения лабораторной работы**

1. **Проектирование диаграммы компонентов**



**2. Реализация диаграммы:**

import java.util.List;

interface IUserService {

User register(String username, String password);

User login(String username, String password);

}

interface IProductService {

List<Product> getProducts();

Product addProduct(Product product);

}

interface IOrderService {

Order createOrder(int userId, List<Integer> productIds);

Order getOrderStatus(int orderId);

}

interface IPaymentService {

boolean processPayment(int orderId, double amount);

}

interface INotificationService {

void sendNotification(int userId, String message);

}

import java.util.HashMap;

import java.util.Map;

public class UserService implements IUserService {

private Map<String, User> users = new HashMap<>();

@Override

public User register(String username, String password) {

User user = new User(username, password);

users.put(username, user);

return user;

}

@Override

public User login(String username, String password) {

User user = users.get(username);

if (user != null && user.getPassword().equals(password)) {

return user;

} else {

throw new IllegalArgumentException("Invalid username or password");

}

}

}

import java.util.ArrayList;

import java.util.List;

public class ProductService implements IProductService {

private List<Product> products = new ArrayList<>();

@Override

public List<Product> getProducts() {

return products;

}

@Override

public Product addProduct(Product product) {

products.add(product);

return product;

}

}

import java.util.List;

import java.util.stream.Collectors;

public class OrderService implements IOrderService {

private final IProductService productService;

private final IPaymentService paymentService;

private final INotificationService notificationService;

public OrderService(IProductService productService, IPaymentService paymentService, INotificationService notificationService) {

this.productService = productService;

this.paymentService = paymentService;

this.notificationService = notificationService;

}

@Override

public Order createOrder(int userId, List<Integer> productIds) {

List<Product> products = productService.getProducts()

.stream()

.filter(p -> productIds.contains(p.getId()))

.collect(Collectors.toList());

if (products.isEmpty()) {

throw new RuntimeException("Selected products not found.");

}

Order order = new Order(userId, products);

double totalAmount = products.stream().mapToDouble(Product::getPrice).sum();

if (paymentService.processPayment(order.getId(), totalAmount)) {

order.setStatus("Paid");

notificationService.sendNotification(userId, "Your order has been successfully paid.");

} else {

order.setStatus("Payment Failed");

notificationService.sendNotification(userId, "Payment failed. Please try again.");

}

return order;

}

@Override

public Order getOrderStatus(int orderId) {

return new Order(orderId, "In Progress");

}

}

public class PaymentService implements IPaymentService {

@Override

public boolean processPayment(int orderId, double amount) {

System.out.println("Processing payment for order " + orderId + " with amount " + amount);

return true;

}

}

public class NotificationService implements INotificationService {

@Override

public void sendNotification(int userId, String message) {

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

}

}

class User {

private String username;

private String password;

public User(String username, String password) {

this.username = username;

this.password = password;

}

public String getPassword() {

return password;

}

}

class Product {

private int id;

private String name;

private double price;

public Product(int id, String name, double price) {

this.id = id;

this.name = name;

this.price = price;

}

public int getId() {

return id;

}

public double getPrice() {

return price;

}

}

class Order {

private static int counter = 1;

private int id;

private int userId;

private List<Product> products;

private String status;

public Order(int userId, List<Product> products) {

this.id = counter++;

this.userId = userId;

this.products = products;

this.status = "Created";

}

public Order(int id, String status) {

this.id = id;

this.status = status;

}

public int getId() {

return id;

}

public void setStatus(String status) {

this.status = status;

}

}

import java.util.List;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

IUserService userService = new UserService();

IProductService productService = new ProductService();

IPaymentService paymentService = new PaymentService();

INotificationService notificationService = new NotificationService();

IOrderService orderService = new OrderService(productService, paymentService, notificationService);

Scanner scanner = new Scanner(System.in);

System.out.println("Welcome to Order Management System!");

System.out.println("Register user:");

System.out.print("Enter username: ");

String username = scanner.next();

System.out.print("Enter password: ");

String password = scanner.next();

User user = userService.register(username, password);

productService.addProduct(new Product(1, "Laptop", 1200.0));

productService.addProduct(new Product(2, "Phone", 800.0));

System.out.println("Creating order...");

Order order = orderService.createOrder(user.getId(), List.of(1, 2));

System.out.println("Order status: " + order.getStatus());

scanner.close();

}

}