**Курс:** Шаблоны проектирования приложений

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

**Лабораторная работа: Проектирование и реализация системы управления библиотекой с использованием диаграммы классов на языке C#**

**Цель работы:**

Изучить принципы построения диаграммы классов и применить их для разработки системы управления библиотекой с последующей реализацией на языке C#.

**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:**

Создайте диаграмму классов, которая включает основные сущности системы и их взаимосвязи. Основные классы:

* **Book** (Книга)
* **Reader** (Читатель)
* **Librarian** (Библиотекарь)
* **Loan** (Выдача)

Каждый класс должен содержать необходимые свойства и методы для выполнения задач.

**Пример:**

* Класс **Book**:
  + Свойства: Title, Author, ISBN, IsAvailable.
  + Методы: MarkAsLoaned(), MarkAsAvailable().
* Класс **Reader**:
  + Свойства: Id, Name, Email.
  + Методы: BorrowBook(), ReturnBook().
* Класс **Librarian**:
  + Свойства: Id, Name, Position.
  + Методы: AddBook(), RemoveBook().
* Класс **Loan**:
  + Свойства: Book, Reader, LoanDate, ReturnDate.
  + Методы: IssueLoan(), CompleteLoan().

**2. Реализация диаграммы классов на языке C#:**

На основе диаграммы классов необходимо реализовать код программы на языке C#.

**Требования к отчету:**

1. **Диаграмма классов** с описанием основных классов и их взаимодействий.
2. **Описание системы**: краткое описание классов и их методов.
3. **Исходный код**: реализация классов на языке C#.
4. **Примеры работы программы**: примеры использования системы.
5. **Выводы**: анализ выполнения работы и достижения целей.

**Дополнительные задания (по желанию):**

1. Реализовать возможность добавления и удаления читателей.
2. Добавить функцию поиска книг по автору или названию.
3. Добавить отчеты для библиотекаря: список занятых и доступных книг.
4. Реализовать сохранение данных о книгах и читателях в файл или базу данных.

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

**Описание задачи:**  
В этой лабораторной работе вы создадите систему для управления заказами, которая состоит из нескольких компонентов: ProductService, OrderService, PaymentService, NotificationService и UserService. Ваша задача – спроектировать диаграмму компонентов для этой системы и реализовать основные классы и интерфейсы на языке C#.

**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**
   * Создайте диаграмму компонентов, где будут отображены все перечисленные компоненты.
   * Определите интерфейсы для взаимодействия между компонентами. Например:
     + IUserService для компонента UserService
     + IProductService для компонента ProductService
     + IOrderService для компонента OrderService
     + IPaymentService для компонента PaymentService
     + INotificationService для компонента NotificationService
   * Отобразите связи между компонентами на диаграмме, указывая зависимости. Например, OrderService должен зависеть от ProductService и PaymentService.

public interface IUserService

{

User Register(string username, string password);

User Login(string username, string password);

}

public interface IProductService

{

List<Product> GetProducts();

Product AddProduct(Product product);

}

public interface IOrderService

{

Order CreateOrder(int userId, List<int> productIds);

Order GetOrderStatus(int orderId);

}

public interface IPaymentService

{

bool ProcessPayment(int orderId, decimal amount);

}

public interface INotificationService

{

void SendNotification(int userId, string message);

}

Пример реализации компонента OrderService:

public class OrderService : IOrderService

{

private readonly IProductService \_productService;

private readonly IPaymentService \_paymentService;

private readonly INotificationService \_notificationService;

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

{

\_productService = productService;

\_paymentService = paymentService;

\_notificationService = notificationService;

}

public Order CreateOrder(int userId, List<int> productIds)

{

// Проверка наличия продуктов

var products = \_productService.GetProducts().Where(p => productIds.Contains(p.Id)).ToList();

if (!products.Any())

{

throw new Exception("Выбранные товары не найдены.");

}

// Создание заказа

var order = new Order { UserId = userId, Products = products, Status = "Created" };

// Обработка платежа

decimal totalAmount = products.Sum(p => p.Price);

if (\_paymentService.ProcessPayment(order.Id, totalAmount))

{

order.Status = "Paid";

\_notificationService.SendNotification(userId, "Ваш заказ успешно оплачен.");

}

else

{

order.Status = "Payment Failed";

\_notificationService.SendNotification(userId, "Платеж не прошел. Попробуйте снова.");

}

return order;

}

public Order GetOrderStatus(int orderId)

{

// Возвращает статус заказа

// Для простоты, здесь можно вернуть фиктивный заказ

return new Order { Id = orderId, Status = "In Progress" };

}

}

**Ожидаемые результаты**

1. **Диаграмма компонентов** с указанием интерфейсов и связей между ними.
2. **Код C#** с реализацией всех компонентов и интерфейсов, а также консольный интерфейс для пользователя.
3. **Отчет**, содержащий описание диаграммы компонентов, код всех компонентов и взаимодействие между ними.

**Итог:**  
Выполнив данную лабораторную работу, вы изучите, как проектировать и реализовывать многокомпонентную архитектуру приложения на C#, а также как использовать диаграмму компонентов для представления архитектуры системы.