Практическая работа № 2.  
шаблон проектирования  
Model-view-controller (MVC)

**Цель практической работы**

Закрепление теоретических знаний по основам применения шаблона проектирования Model-View-Controller при разработке приложений.

**Постановка задачи**

Паттерны проектирования — это проверенные решения распространенных проблем, с которыми разработчики сталкиваются при проектировании программного обеспечения. Они помогают создать гибкую и поддерживаемую архитектуру, улучшая читаемость и повторное использование кода.

Паттерн проектирования MVC (Model-View-Controller) является одним из наиболее распространенных подходов в разработке программного обеспечения. Он разделяет приложение на три основные компоненты: модель, представление и контроллер. Это разделение помогает улучшить организацию кода, облегчить его тестирование и упростить поддержку.

**Структура паттерна**

Паттерн "Стратегия" состоит из следующих компонентов:

1. **Контекст (Context)**: класс, который использует стратегию. Он содержит ссылку на объект стратегии и делегирует выполнение алгоритма этому объекту.
2. **Стратегия (Strategy)**: интерфейс, который определяет общий контракт для всех конкретных стратегий.
3. **Конкретные стратегии (ConcreteStrategy)**: классы, которые реализуют интерфейс стратегии и предоставляют конкретные алгоритмы.

**Пример проекта Windows Forms с использованием MVC**

**Описание проекта**

В этом примере мы создадим простое приложение для управления списком контактов. Приложение будет позволять пользователю добавлять, удалять и отображать контакты.

**Структура проекта**

1. **Model**: класс Contact, который будет представлять контакт.
2. **View**: форма MainForm, которая будет отображать интерфейс пользователя.
3. **Controller**: класс ContactController, который будет управлять взаимодействием между моделью и представлением.

**Шаг 1: Создание модели**

Создадим класс Contact в папке Model проекта.

// Model/Contact.cs

public class Contact

{

    public string Name { get; set; }

    public string PhoneNumber { get; set; }

    public Contact(string name, string phoneNumber)

    {

        Name = name;

        PhoneNumber = phoneNumber;

    }

}

**Шаг 2: Определение интерфейсов**

Создадим интерфейсы для представления и контроллера.

// Interfaces/IContactView.cs

using System.Collections.Generic;

public interface IContactView

{

    event Action<string, string> ContactAdded;

    event Action<string> ContactRemoved;

    void UpdateContactList(List<Contact> contacts);

}

// Interfaces/IContactController.cs

public interface IContactController

{

    void AddContact(string name, string phoneNumber);

    void RemoveContact(string name);

}

**Шаг 3: Создание представления**

Создадим форму MainForm, которая будет содержать элементы управления для отображения и управления контактами.

// View/MainForm.cs

public partial class MainForm : Form, IContactView

{

    public event Action<string, string> ContactAdded;

    public event Action<string> ContactRemoved;

    public MainForm()

    {

        InitializeComponent();

    }

    public void UpdateContactList(List<Contact> contacts)

    {

        listBoxContacts.Items.Clear();

        foreach (var contact in contacts)

        {

            listBoxContacts.Items.Add($"{contact.Name} - {contact.PhoneNumber}");

        }

    }

    private void buttonAdd\_Click(object sender, EventArgs e)

    {

        ContactAdded?.Invoke(textBoxName.Text, textBoxPhoneNumber.Text);

    }

    private void buttonRemove\_Click(object sender, EventArgs e)

    {

        if (listBoxContacts.SelectedItem != null)

        {

            string selectedContact = listBoxContacts.SelectedItem.ToString();

            string name = selectedContact.Split('-')[0].Trim();

            ContactRemoved?.Invoke(name);

        }

    }

}

**Шаг 4: Создание контроллера**

Создадим класс контроллера, который будет управлять логикой приложения.

// Controller/ContactController.cs

public class ContactController : IContactController

{

    private List<Contact> contacts;

    private IContactView view;

    public ContactController(IContactView view)

    {

        this.view = view;

        contacts = new List<Contact>();

        this.view.ContactAdded += OnContactAdded;

        this.view.ContactRemoved += OnContactRemoved;

    }

    private void OnContactAdded(string name, string phoneNumber)

    {

        AddContact(name, phoneNumber);

    }

    private void OnContactRemoved(string name)

    {

        RemoveContact(name);

    }

    public void AddContact(string name, string phoneNumber)

    {

        contacts.Add(new Contact(name, phoneNumber));

        view.UpdateContactList(contacts);

    }

    public void RemoveContact(string name)

    {

        contacts.RemoveAll(c => c.Name == name);

        view.UpdateContactList(contacts);

    }

}

**Шаг 5: Связывание компонентов в Main**

Теперь свяжем все компоненты в методе Main.

// Program.cs

static class Program

{

    [STAThread]

    static void Main()

    {

        Application.EnableVisualStyles();

        Application.SetCompatibleTextRenderingDefault(false);

        IContactView view = new MainForm();

        IContactController controller = new ContactController(view);

        Application.Run((Form)view);

    }

}

**Задание на практическую работу**

1. В соответствии с паттерном «**MVC**» разработать Windows Forms (.Net 8) приложение согласно варианту.
2. Предусмотреть создание и отображение на форме объектов согласно варианту.
3. Для отображения созданных объектов использовать DataGridView.
4. Для успешной защиты:

* программа должна быть задокументирована с помощью комментариев,
* код программы не должен содержать не используемые блоки, должен быть «чистым» и стилистически верно оформлен,
* интерфейс программы должен быть гибким и удобным.

**Варианты заданий на практическую работу**

**1. Управление библиотекой**

* **Model:** Класс Book, который содержит свойства книги (название, автор, год издания и т.д.), а также класс Library, который управляет списком книг.
* **View:** Формы для отображения списка книг, добавления/редактирования деталей книги, а также кнопки для выполнения действий (добавить, удалить, редактировать).
* **Controller:** Логика для обработки событий от View, взаимодействие с Model (добавление/удаление книг) и обновление View при изменениях.

**2. Калькулятор**

* **Model:** Класс Calculator, который реализует логику вычислений (методы для сложения, вычитания, умножения и деления).
* **View:** Формы с кнопками для чисел и операций, а также текстовое поле для отображения результата.
* **Controller:** Обработка событий нажатий кнопок, вызов методов Model для вычислений и обновление View с результатом.

**3. Список задач (To-Do List)**

* **Model:** Класс Task, который содержит свойства задачи (название, дата, статус), и класс TaskList, который управляет коллекцией задач.
* **View:** Формы для отображения списка задач, добавления новых задач и отметки задач как выполненных.
* **Controller:** Логика для обработки событий (добавление/удаление задач), взаимодействие с Model и обновление View.

**4. Управление контактами**

* **Model:** Класс Contact, который содержит информацию о контакте (имя, номер телефона, адрес), и класс ContactList, который управляет коллекцией контактов.
* **View:** Формы для отображения списка контактов, добавления/редактирования контактов и поиска.
* **Controller:** Обработка событий (добавление, редактирование, удаление контактов) и обновление View.

**5. Учебный план**

* **Model:** Класс Course, который содержит информацию о курсе (название, код, описание), и класс Curriculum, который управляет списком курсов.
* **View:** Формы для отображения списка курсов, добавления и удаления курсов.
* **Controller:** Логика для обработки событий (добавление/удаление курсов) и обновление View.

**6. Приложение для учета расходов**

* **Model:** Класс Expense, который содержит информацию о расходах (сумма, категория, дата), и класс ExpenseTracker, который управляет списком расходов.
* **View:** Формы для ввода и отображения расходов, а также отчетов.
* **Controller:** Обработка событий (добавление/удаление расходов), взаимодействие с Model и обновление View.

**7. Галерея изображений**

* **Model:** Класс ImageItem, который содержит информацию об изображении (путь, описание), и класс ImageGallery, который управляет коллекцией изображений.
* **View:** Формы для отображения изображений, кнопки для загрузки и удаления изображений.
* **Controller:** Логика для обработки событий (загрузка/удаление изображений), взаимодействие с Model и обновление View.

**8. Система управления запасами**

* **Model:** Класс Product, который содержит информацию о товаре (название, количество, цена), и класс Inventory, который управляет списком товаров.
* **View:** Формы для отображения списка товаров, добавления/редактирования товаров.
* **Controller:** Обработка событий (добавление/удаление товаров), взаимодействие с Model и обновление View.

**9. Приложение для заметок**

* **Model:** Класс Note, который содержит текст заметки и дату, и класс NoteList, который управляет коллекцией заметок.
* **View:** Формы для отображения списка заметок, добавления и редактирования заметок.
* **Controller:** Логика для обработки событий (добавление/редактирование заметок) и обновление View.

**10. Приложение для бронирования мест**

* **Model:** Класс Seat, который содержит информацию о месте (номер, статус), и класс BookingSystem, который управляет списком мест.
* **View:** Формы для отображения доступных мест, выбора мест для бронирования.
* **Controller:** Обработка событий (бронирование/отмена мест), взаимодействие с Model и обновление View.

**11. Календарь событий**

* **Model:** Класс Event, который содержит информацию о событии (название, дата, время), и класс EventCalendar, который управляет списком событий.
* **View:** Формы для отображения событий, добавления и редактирования событий.
* **Controller:** Логика для обработки событий (добавление/редактирование событий) и обновление View.

**12. Управление рецептами**

* **Model:** Класс Recipe, который содержит информацию о рецепте (название, ингредиенты, инструкции), и класс RecipeBook, который управляет коллекцией рецептов.
* **View:** Формы для отображения списка рецептов, добавления и редактирования рецептов.
* **Controller:** Обработка событий (добавление/редактирование рецептов) и обновление View.

**13. Приложение для опросов**

* **Model:** Класс Question, который содержит текст вопроса и варианты ответов, и класс Survey, который управляет списком вопросов.
* **View:** Формы для создания и отображения опросов, кнопки для отправки ответов.
* **Controller:** Логика для обработки событий (создание/отправка опроса) и обновление View.

**14. Система регистрации пользователей**

* **Model:** Класс User , который содержит информацию о пользователе (имя, пароль, email), и класс User Manager, который управляет списком пользователей.
* **View:** Формы для регистрации, авторизации и восстановления пароля.
* **Controller:** Обработка событий (регистрация, авторизация) и взаимодействие с Model.

**15. Приложение для управления проектами**

* **Model:** Класс Project, который содержит информацию о проекте (название, описание, статус), и класс ProjectManager, который управляет списком проектов.
* **View:** Формы для отображения списка проектов, добавления и редактирования проектов.
* **Controller:** Логика для обработки событий (добавление/редактирование проектов) и обновление View.

**16. Приложение для генерации отчетов**

* **Model:** Класс ReportData, который содержит информацию для отчета, и класс ReportGenerator, который управляет логикой генерации отчетов.
* **View:** Формы для ввода данных и отображения сгенерированных отчетов.
* **Controller:** Обработка событий (ввод данных, генерация отчетов) и обновление View.

**17. Система управления школой**

* **Model:** Класс Student, Teacher, Subject, который содержит информацию о студентах, учителях и предметах, и класс SchoolManager, который управляет всеми данными.
* **View:** Формы для отображения списка учеников, учителей и предметов, добавления и редактирования.
* **Controller:** Логика для обработки событий (добавление/редактирование данных) и обновление View.

**18. Приложение для управления рецептами блюд**

* **Model:** Класс DishRecipe, который содержит информацию о блюде (название, ингредиенты, инструкции), и класс RecipeManager, который управляет коллекцией рецептов блюд.
* **View:** Формы для отображения списка рецептов, добавления и редактирования.
* **Controller:** Обработка событий (добавление/редактирование рецептов) и обновление View.

**19. Приложение для ведения дневника**

* **Model:** Класс DiaryEntry, который содержит текст записи и дату, и класс Diary, который управляет коллекцией записей.
* **View:** Формы для отображения списка записей, добавления и редактирования записей.
* **Controller:** Логика для обработки событий (добавление/редактирование записей) и обновление View.

**20. Система управления событиями**

* **Model:** Класс Event, который содержит информацию о мероприятии (название, дата, место), и класс EventManager, который управляет списком мероприятий.
* **View:** Формы для отображения списка мероприятий, добавления и редактирования.
* **Controller:** Обработка событий (добавление/редактирование мероприятий) и обновление View.

# Содержание пояснительной записки

1. Постановка задачи. Приводится теоретический материал, использованный при написании приложения.

2. Формулировка задания и вариант. Приводится задание на лабораторную работу и вариант этого задания.

3. Описание выполняемых действий. Необходимо привести описание последовательности разработки программы, реализации используемых методов, алгоритмов, блок-схем.

4. Анализ результатов. Привести анализ входных и выходных данных. Показать результаты выполнения программного кода. Предоставить скриншоты обработки тестовых примеров. Сделать выводы.

5. Листинг программы. Привести листинг разработанного программного кода, содержание файлов входных и выходных данных.

# Используемое программное обеспечение

1. Среда программирования MS Visual Studio Community 2022 (Свободно распространяемое программное обеспечение (в учебных целях));
2. Microsoft Office Standard 2007 (Open License: 42267924);
3. Open Office (Свободно распространяемое программное обеспечение).
4. Браузер (Свободно распространяемое программное обеспечение).

# Список литературы

* + - 1. Мейер Б. Объектно-ориентированное программирование и программная инженерия [Электронный ресурс]/ Мейер Б. – Электрон. текстовые данные. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. – 285 c.
      2. Биллиг, В. A. Основы объектного программирования на С# (C# 3.0, Visual Studio 2008) [Электронный ресурс] : учебное пособие / В. A. Биллиг. — Электрон. текстовые данные. — Москва, Саратов : Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2017. — 583 c. — 978-5-4487-0145-0. — Режим доступа: http://www.iprbookshop.ru/72339.html
      3. Павловская, Т. А. Программирование на языке высокого уровня C# [Электронный ресурс] / Т. А. Павловская. — 2-е изд. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 245 c. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/73713.html
      4. Агапов, В. П. Основы программирования на языке С# [Электронный ресурс] : учебное пособие / В. П. Агапов. — Электрон. текстовые данные. — М. : Московский государственный строительный университет, ЭБС АСВ, 2012. — 128 c. — 978-5-7264-0576-6. — Режим доступа: http://www.iprbookshop.ru/16366.html
      5. Медведев, М. А. Программирование на СИ# [Электронный ресурс] : учебное пособие / М. А. Медведев, А. Н. Медведев ; под ред. А. В. Присяжный. — Электрон. текстовые данные. — Екатеринбург : Уральский федеральный университет, ЭБС АСВ, 2015. — 64 c. — 978-5-7996-1561-1. — Режим доступа: http://www.iprbookshop.ru/69667.html
      6. Казанский А.А. Объектно-ориентированное программирование на языке Microsoft Visual С# в среде разработки Microsoft Visual Studio 2008 и .NET Framework. 4.3 [Электронный ресурс]: учебное пособие и практикум/ Казанский А.А.— Электрон. текстовые данные.— М.: Московский государственный строительный университет, ЭБС АСВ, 2011.— 180 c
      7. Уйманова Н.А. Основы объектно-ориентированного программирования [Электронный ресурс]: практикум/ Уйманова Н.А., Таспаева М.Г.— Электрон. текстовые данные.— Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2017.— 156 c.— Режим доступа: http://www.iprbookshop.ru/78808.html.— ЭБС «IPRbooks»
      8. Новиков П.В. Объектно-ориентированное программирование [Электронный ресурс]: учебно-методическое пособие к лабораторным работам/ Новиков П.В.— Электрон. текстовые данные.— Саратов: Вузовское образование, 2017.— 124 c.— Режим доступа: http://www.iprbookshop.ru/64650.html.— ЭБС «IPRbooks»