

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
КАФЕДРА ИИТ

Лабораторная работа №6

По дисциплине: «Современные платформы программирования»

Выполнил:

Студент 3 курса

группы ПО-8:

Печко В.И.

Проверил:

Крощенко А.А.

Цель работы: приобрести навыки применения паттернов проектирования при решении практических задач с использованием языка C#.

Вариант 18

Задание 1.

8) Торговый автомат с возможностью выдачи любого выбранного товара (шоколадные батончики, чипсы, пакетированные соки и т.д.)

Для реализации данной задачи был выбран паттерн "Фабричный метод". Этот паттерн обеспечивает создание объектов без явного указания их конкретных классов, делегируя процесс создания специализированным подклассам. В данном контексте абстрактный класс `ProductFactory` выступает в роли "Фабрики", предоставляя абстрактный метод `CreateProduct`, который подклассы реализуют для создания конкретных продуктов (`ChocolateBar`, `Chips`, `PackagedJuice`).

Код программы:

Program.cs:

```
public abstract class Product
{
    public string Name { get; protected set; }
    public decimal Price { get; protected set; }

    public Product(string name, decimal price)
    {
        Name = name;
        Price = price;
    }
}

public class ChocolateBar : Product
{
    public ChocolateBar(string name, decimal price) : base(name, price)
    {
    }
}

public class Chips : Product
{
    public Chips(string name, decimal price) : base(name, price)
    {
    }
}

public class PackagedJuice : Product
{
    public PackagedJuice(string name, decimal price) : base(name, price)
    {
    }
}
```

```

    }

    public abstract class ProductFactory
    {
        public abstract Product CreateProduct(string name, decimal price);
    }

    public class ChocolateBarFactory : ProductFactory
    {
        public override Product CreateProduct(string name, decimal price)
        {
            return new ChocolateBar(name, price);
        }
    }

    public class ChipsFactory : ProductFactory
    {
        public override Product CreateProduct(string name, decimal price)
        {
            return new Chips(name, price);
        }
    }

    public class PackagedJuiceFactory : ProductFactory
    {
        public override Product CreateProduct(string name, decimal price)
        {
            return new PackagedJuice(name, price);
        }
    }

    public class VendingMachine
    {
        private List<Product> products = new List<Product>();

        public void AddProduct(Product product)
        {
            products.Add(product);
        }

        public decimal CalculateTotal()
        {
            decimal total = 0;
            foreach (var product in products)
            {
                total += product.Price;
            }
            return total;
        }
    }

    class Program

```

```

{
    static void Main(string[] args)
    {
        VendingMachine vendingMachine = new VendingMachine();
        ProductFactory chocolateBarFactory = new ChocolateBarFactory();
        ProductFactory chipsFactory = new ChipsFactory();
        ProductFactory packagedJuiceFactory = new PackagedJuiceFactory();

        Product chocolateBar = chocolateBarFactory.CreateProduct("Milk Chocolate", 1.99m);
        Product chips = chipsFactory.CreateProduct("Potato Chips", 2.99m);
        Product packagedJuice = packagedJuiceFactory.CreateProduct("Orange Juice", 3.99m);

        vendingMachine.AddProduct(chocolateBar);
        vendingMachine.AddProduct(chips);
        vendingMachine.AddProduct(packagedJuice);
        Console.WriteLine("Total: $" + vendingMachine.CalculateTotal());
    }
}

```

Результат программы:

Total: \$8,97

Задание 2.

8) ДУ телевизора. Реализовать иерархию телевизоров для конкретных производителей и иерархию средств дистанционного управления. Телевизоры должны иметь присущие им атрибуты и функции. ДУ имеет набор функций для изменения текущего канала, увеличения/уменьшения громкости, включения/выключения телевизора и т.д. Эти функции должны отличаться для различных устройств ДУ.

Для данной реализации был выбран паттерн "Мост" (Bridge). Этот паттерн позволяет разделять абстракцию и реализацию таким образом, чтобы они могли изменяться независимо друг от друга. Интерфейс `IRemote` и классы `Remote` и `AdvancedRemote` представляют собой абстракцию, т.е., способ управления телевизором. С другой стороны, интерфейс `ITV` и его реализации `SamsungTV` и `LGTv` представляют собой реализацию, т.е., сами телевизоры.

Код программы:

Program.cs:

```

using System;

public interface IRemote
{
    void Power();
    void VolumeUp();
    void VolumeDown();
    void ChannelUp();
    void ChannelDown();
}

```

```

public interface ITV
{
    void TurnOn();
    void TurnOff();
    void SetVolume(int percent);
    void SetChannel(int number);
}

public class Remote : IRemote
{
    protected ITV tv;

    public Remote(ITV tv)
    {
        this.tv = tv;
    }

    public void Power()
    {
        tv.TurnOn();
    }

    public void VolumeUp()
    {
        tv.SetVolume(10);
    }

    public void VolumeDown()
    {
        tv.SetVolume(-10);
    }

    public void ChannelUp()
    {
        tv.SetChannel(1);
    }

    public void ChannelDown()
    {
        tv.SetChannel(-1);
    }
}

public class AdvancedRemote : Remote
{
    public AdvancedRemote(ITV tv) : base(tv)
    {
    }

    public void Mute()
    {

```

```

        tv.SetVolume(0);
    }
}

public class SamsungTV : ITV
{
    public void TurnOn()
    {
        Console.WriteLine("Samsung TV is turned on.");
    }

    public void TurnOff()
    {
        Console.WriteLine("Samsung TV is turned off.");
    }

    public void SetVolume(int percent)
    {
        Console.WriteLine($"Samsung TV volume is set to {percent}%");
    }

    public void SetChannel(int number)
    {
        Console.WriteLine($"Samsung TV channel is set to {number}");
    }
}

public class LGTV : ITV
{
    public void TurnOn()
    {
        Console.WriteLine("LG TV is turned on.");
    }

    public void TurnOff()
    {
        Console.WriteLine("LG TV is turned off.");
    }

    public void SetVolume(int percent)
    {
        Console.WriteLine($"LG TV volume is set to {percent}%");
    }

    public void SetChannel(int number)
    {
        Console.WriteLine($"LG TV channel is set to {number}");
    }
}

class Program
{

```

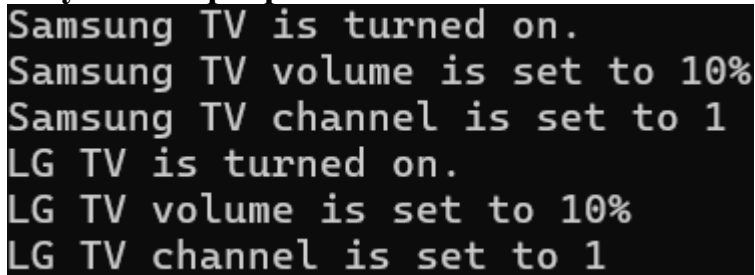
```

static void Main(string[] args)
{
    ITV samsung = new SamsungTV();
    IRemote remote = new Remote(samsung);
    remote.Power();
    remote.VolumeUp();
    remote.ChannelUp();

    ITV lg = new LGTV();
    IRemote advancedRemote = new AdvancedRemote(lg);
    advancedRemote.Power();
    advancedRemote.VolumeUp();
    advancedRemote.ChannelUp();
}
}

```

Результат программы:



```

Samsung TV is turned on.
Samsung TV volume is set to 10%
Samsung TV channel is set to 1
LG TV is turned on.
LG TV volume is set to 10%
LG TV channel is set to 1

```

Задание 3.

8) Вспомогательная библиотека для работы с текстовыми файлами. Должны быть предусмотрены следующие функции: чтение одного или нескольких файлов, изменение файла(ов), отмена последней выполненной операции (одной в случае простой единичной операции или нескольких в случае сложной операции), последовательное выполнение нескольких операций.

Для данной реализации был выбран паттерн "Команда" (Command). Этот паттерн представляет собой поведенческий шаблон проектирования, который инкапсулирует запрос как объект, позволяя настраивать параметры вызова операций, ставить запросы в очередь или вести их журналирование, а также поддерживать отмену операций. Интерфейс `ICommand` определяет методы `Execute()` и `UnExecute()`, которые выполняют и отменяют операции соответственно. Классы `ReadFileCommand` и `WriteFileCommand` реализуют этот интерфейс, представляя операции чтения и записи файла. Класс `CommandInvoker` представляет объект, который хранит историю выполненных команд и позволяет выполнить или отменить их.

Код программы:

Program.cs:

```

public interface ICommand
{
    void Execute();
}

```

```

    void UnExecute();
}

public class ReadFileCommand : ICommand
{
    private string filePath;

    public ReadFileCommand(string filePath)
    {
        this.filePath = filePath;
    }

    public void Execute()
    {
        string text = File.ReadAllText(filePath);
        Console.WriteLine(text);
    }

    public void UnExecute()
    {
        // Не применимо для операции чтения
    }
}

public class WriteFileCommand : ICommand
{
    private string filePath;
    private string text;
    private string backup;

    public WriteFileCommand(string filePath, string text)
    {
        this.filePath = filePath;
        this.text = text;
    }

    public void Execute()
    {
        backup = File.ReadAllText(filePath);
        File.WriteAllText(filePath, text);
    }

    public void UnExecute()
    {
        File.WriteAllText(filePath, backup);
    }
}

public class CommandInvoker
{
    private Stack<ICommand> commandHistory = new Stack<ICommand>();

```



```

public void ExecuteCommand(ICommand command)
{
    command.Execute();
    commandHistory.Push(command);
}

public void UndoLastCommand()
{
    if (commandHistory.Count > 0)
    {
        ICommand command = commandHistory.Pop();
        command.UnExecute();
    }
}

class Program
{
    static void Main(string[] args)
    {
        CommandInvoker invoker = new CommandInvoker();
        ICommand readFileCommand = new ReadFileCommand("text.txt");
        ICommand writeFileCommand = new WriteFileCommand("text.txt", "Hello, World!");

        invoker.ExecuteCommand(readFileCommand);
        invoker.ExecuteCommand(writeFileCommand);
        invoker.UndoLastCommand();    }}

```

Результат программы:

```

Hello, World!
This is a test file.
You can add any text you need here.
For example, this could be information, data, or anything else.
Good luck!

```

Вывод: приобрели навыки применения паттернов проектирования при решении практических задач с использованием языка C#.