Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №6
По дисциплине: «СПП»

Выполнил:
Студент 3 курса
Группы ПО-3
Новикевич А.А.
Проверил:
Монтик Н.С.

2020

Лабораторная работа №6

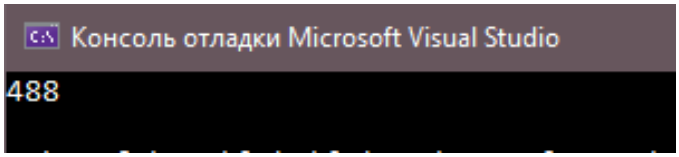
Цель работы: приобрести навыки применения паттернов проектирования при решении практических задач с использованием языка C#.

Задание 1: Проект «Туристическое бюро». Реализовать возможность выбора программы тура (проезд, проживание, питание, посещение музеев, выставок, экскурсии и т.д.). Должна формироваться итоговая стоимость заказа.

Код программы:

```
using System;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            Services services = Services.CreateBuilder().Travel("Brest",
131).Accomodation("hotel", 15).
                Nutrition("all included", 100).MusiemsVisiting("all", 242);
            Console.WriteLine(services.SumCost());
        }
    }
}

public class Services
{
    public Service Travel { get; set; }
    public Service Accomodation { get; set; }
    public Service Nutrition { get; set; }
    public Service MusiemsVisiting { get; set; }
    public Service ExhibitionsVisiting { get; set; }
    public Service ExcursionsVisiting { get; set; }

    public int SumCost()
    {
        int result = 0;
        result += Travel?.Cost ?? 0;
        result += Accomodation?.Cost ?? 0;
        result += Nutrition?.Cost ?? 0;
        result += MusiemsVisiting?.Cost ?? 0;
        result += ExhibitionsVisiting?.Cost ?? 0;
        result += ExcursionsVisiting?.Cost ?? 0;
        return result;
    }

    public static ServiceBuilder CreateBuilder()
    {
        return new ServiceBuilder();
    }
}

public class Service
{
    public string Name { get; set; }
    public int Cost { get; set; }

    public Service(string name, int cost)
    {
```

```
            Name = name;
            Cost = cost;
        }
    }

    public class ServiceBuilder
    {
        private Services _services;
        public ServiceBuilder()
        {
            _services = new Services();
        }
        public ServiceBuilder Travel(string name, int cost)
        {
            _services.Travel = new Service(name, cost);
            return this;
        }
        public ServiceBuilder Accomodation(string name, int cost)
        {
            _services.Accomodation = new Service(name, cost);
            return this;
        }
        public ServiceBuilder Nutrition(string name, int cost)
        {
            _services.Nutrition = new Service(name, cost);
            return this;
        }
        public ServiceBuilder MusiemsVisiting(string name, int cost)
        {
            _services.MusiemsVisiting = new Service(name, cost);
            return this;
        }
        public ServiceBuilder ExhibitionsVisiting(string name, int cost)
        {
            _services.ExhibitionsVisiting = new Service(name, cost);
            return this;
        }
        public ServiceBuilder ExcursionsVisiting(string name, int cost)
        {
            _services.ExcursionsVisiting = new Service(name, cost);
            return this;
        }

        public static implicit operator Services(ServiceBuilder builder)
        {
            return builder._services;
        }
    }
```



Задание 2:

Проект «Файловая система». Реализуйте модель работы файловой системы. Должна поддерживаться иерархичность ФС на уровне директорий и отдельных файлов. Файлы могут иметь все основные присущие им атрибуты (размер, расширение, дата создания т.д.

```
using System;
using System.Collections.Generic;

namespace ConsoleApp2
{
    class Program
    {
        static void Main(string[] args)
        {
            ExplorerComponent folder1 = new Directory("Folder1");
            ExplorerComponent folder2 = new Directory("folder2");
            ExplorerComponent folder3 = new Directory("folder3");

            ExplorerComponent text = new File("text", DateTime.Now +
TimeSpan.FromTicks(TimeSpan.TicksPerDay), 1000, "txt");
            ExplorerComponent exe = new File("pe", DateTime.Now +
TimeSpan.FromTicks(TimeSpan.TicksPerDay * 2), 4356, "exe");
            ExplorerComponent png = new File("photo", DateTime.Now, 1000, "png");

            folder1.Add(folder2);
            folder1.Add(text);
            folder1.Add(exe);

            folder2.Add(folder3);
            folder2.Add(png);
            Console.WriteLine(folder1.GetInfo());
        }
    }
    abstract class ExplorerComponent
    {
        public virtual void Add(ExplorerComponent component) { }
        public virtual void Remove(ExplorerComponent component) { }
        public virtual string GetInfo() { return string.Empty; }
    }

    class Directory : ExplorerComponent
    {
        private List<ExplorerComponent> _explorer;

        public string Name { get; set; }

        public Directory() => _explorer = new List<ExplorerComponent>();
        public Directory(string name) : this() => Name = name;

        public override void Add(ExplorerComponent component) =>
_explorer.Add(component);
        public override void Remove(ExplorerComponent component) =>
_explorer.Remove(component);
        public override string GetInfo() => Name;
    }

    class File : ExplorerComponent
    {
        public string Name { get; set; }
        public DateTime? CreationDate { get; set; }
        public uint Size { get; set; }
        public string Extension { get; set; }

        public File() { }
        public File(string name, DateTime date, uint size, string extension)
        {
            Name = name;
            CreationDate = date;
            Size = size;
```

```
                Extension = extension;
            }

            public override string GetInfo() => $"{Name} - {Extension} - {CreationDate} -
        {Size}";
        }
}
```

Задание 3:

Реализовать вывод ФС из 2-й группы заданий. Вывод файлов/директорий должен осуществляться в случайном порядке. Вывести основные атрибуты каждого файла/директории.

Код программы:

```
        using System;
        using System.Collections;
        using System.Collections.Generic;
        using System.Linq;


        namespace ConsoleApp1
        {
            class Program
            {
                static void Main(string[] args)
                {
                    ExplorerComponent folder1 = new Directory("Folder1");
                    ExplorerComponent folder2 = new Directory("folder2");
                    ExplorerComponent folder3 = new Directory("folder3");

                    ExplorerComponent text = new File("text", DateTime.Now +
        TimeSpan.FromTicks(TimeSpan.TicksPerDay), 1000, "txt");
                    ExplorerComponent exe = new File("cp", DateTime.Now +
        TimeSpan.FromTicks(TimeSpan.TicksPerDay * 2), 43, "cpp");
                    ExplorerComponent png = new File("photo", DateTime.Now, 1000, "jpg");


                    folder1.Add(folder2);
                    folder1.Add(text);
                    folder1.Add(exe);

                    folder2.Add(folder3);
                    folder2.Add(png);

                    Iterator<ExplorerComponent> displayer = folder1.GetIterator();


                    while (!displayer.IsDone())
                    {
                        Console.WriteLine(displayer.Next().GetInfo());
                    }
                }
            }

            abstract class ExplorerComponent
            {
                public abstract Iterator<ExplorerComponent> GetIterator();
                public virtual void Add(ExplorerComponent component) { }
                public virtual void Remove(ExplorerComponent component) { }
                public virtual string GetInfo() { return string.Empty; }
```

```
    }

    class Directory : ExplorerComponent
    {
        private List<ExplorerComponent> _items;

        public IReadOnlyCollection<ExplorerComponent> Items { get => _items; }

        public string Name { get; set; }

        public Directory() => _items = new List<ExplorerComponent>();
        public Directory(string name) : this() => Name = name;

        public override void Add(ExplorerComponent component) => _items.Add(component);
        public override void Remove(ExplorerComponent component) =>
_items.Remove(component);
        public override string GetInfo() => Name;

        public override Iterator<ExplorerComponent> GetIterator()
        {
            return new RandomIterator(this);
        }
    }

    class File : ExplorerComponent
    {
        public string Name { get; set; }
        public DateTime? CreationDate { get; set; }
        public uint Size { get; set; }
        public string Extension { get; set; }

        public File() { }
        public File(string name, DateTime date, uint size, string extension)
        {
            Name = name;
            CreationDate = date;
            Size = size;
            Extension = extension;
        }

        public override string GetInfo() => $"{Name} - {Extension} - {CreationDate} -
{Size}";

        public override Iterator<ExplorerComponent> GetIterator()
        {
            return null;
        }
    }

    abstract class Iterator<T>
    {
        public abstract T Next();
        public abstract bool IsDone();
        public abstract T CurrentItem { get; }
    }

    class RandomIterator : Iterator<ExplorerComponent>
    {
        private readonly ExplorerComponent _iterable;
        private ExplorerComponent _current;
        private Directory _currentDirectory;
        private Iterator<ExplorerComponent> _directoryIterator;
        private int[] _indexes;
        private int _currentIndex;
```

```
        public RandomIterator(ExplorerComponent explorerComponent)
        {
            _iterable = explorerComponent;
            _currentDirectory = _iterable as Directory;

            _indexes = new int[_currentDirectory.Items.Count];
            for (int i = 0; i < _currentDirectory.Items.Count; i++)
            {
                _indexes[i] = i;
            }

            Random random = new Random(DateTime.Now.Millisecond);
            _indexes = _indexes.OrderBy(x => random.Next()).ToArray();

            _currentIndex = 0;
        }
        public override ExplorerComponent CurrentItem { get => _current; }


        public override ExplorerComponent Next()
        {
            if (_directoryIterator != null)
            {
                if (!_directoryIterator.IsDone())
                {
                    return _directoryIterator.Next();
                }
                else
                {
                    _directoryIterator = null;
                }
            }

            _current = _currentDirectory.Items.ElementAt(_indexes[_currentIndex]);

            if (_current is Directory)
            {
                _directoryIterator = (_current as Directory).GetIterator();
            }

            _currentIndex += 1;
            return _current;
        }

        public override bool IsDone()
        {
            bool isDone = _directoryIterator != null ? _directoryIterator.IsDone() :
true;
            return _currentIndex >= _indexes.Length && isDone;
        }
    }
}
```

```
cp - cpp - 10.12.2020 21:29:28 - 43
text - txt - 09.12.2020 21:29:28 - 1000
folder2
photo - jpg - 08.12.2020 21:29:28 - 1000
folder3
```

```
folder2
photo - jpg - 08.12.2020 21:29:52 - 1000
folder3
text - txt - 09.12.2020 21:29:52 - 1000
cp - cpp - 10.12.2020 21:29:52 - 43
```