НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп’ютерних систем

ЛАБОРАТОРНА РОБОТА №2

з дисципліни «Об'єктно орієнтоване програмування»

Тема: «Проектування класів із використанням стандартної бібліотеки»

Виконав студент 2 курсу

ФПМ групи КВ-14

Фролов В.П.

Перевірив(-ла):

Київ – 2023

Постановка задачі

Мають бути реалізовані наступні можливості:

- Отримання файлу/папки за повним шляхом (fs.get(name)).
- Отримання файлу/папки за іменем у заданій папці (fs.get(name, folder)).
- Отримання кореневої папки (fs.getRoot()).
- Визначення, чи об'єкт є файлом, чи папкою (obj.isFolder()).
- Отримання атрибутів файлу/папки: ім'я, дата створення, дата модифікації, розмір, для папок кількість вкладних файлів/папок (безпосередньо та загалом) (obj.getName(), obj.getCreationDate(), …).
- Додавання нової папки (fs.createFolder(folder, name)).
- Додавання нового файлу (fs.createFile(folder, name, size)).
- Модифікація файлу (в т.ч. може бути змінено розмір файлу) (fs.modify(file, new_size)).
- Видалення файлу або папки (в т.ч. разом із вкладеними сутностями) (fs.remove(obj)).
- Перейменування файлу/папки (fs.rename(file, new_name)).
- Перенесення файлу/папки до деякої папки (fs.move(file, folder)).
- Копіювання файлу або папки (в т.ч. разом із вкладеними сутностями) (fs.copy(file, folder, [new_name])).

- Пошук всіх вкладених файлів у папці за іменем (для отримання додаткових балів можуть використовуватися не точні імена, а маски в іменах: «?» (довільний символ), «*» (довільна кількість довільних символів)), розміром, датами (дозволити задавати діапазоном). В результаті має бути отримано набір (колекцію) сутностей (find(folder, name_mask), find(folder, min_size, max_size)).
- Вивід структури папки (fs.print(folder)).

Прототипи методів наведено лише для прикладу і можуть бути змінені за бажанням.

Спиратися на використання засобів STL (зокрема, колекцій).

Для отримання максимального балу приділити увагу оптимальній реалізації програми (намагатися за можливості мінімізувати час виконання операцій).

Для отримання додаткових балів:

- Використовувати механізм виключень для обробки помилкових ситуацій (якщо вони можливі).
- Використовувати кешування.
- Реалізувати пошук файлів за маскою.

Код програми

FileSystemExecutor.cs

```
using OOP_L2.FilesSystem;

namespace OOP_L2;

public class Executor
{
   public static void Main()
   {
      var fileSystem = new FileSystem(new FSDirectory("root"));

      fileSystem.AddDirectory(new List<FSDirectory>
      {
         new FSDirectory("dir1"),
         new FSDirectory("dir1"),
         new FSDirectory("dir2")
      });

      fileSystem.AddDirectory(new List<FSDirectory>
      {
         new FSDirectory("dir1.1"),
         new FSDirectory("dir1.2")
      }, "dir1");

      fileSystem.AddFile(new List<FSFile>()
      {
         new FSFile("file1.exe"),
         new FSFile("file2.exe"),
         new FSFile("file3.exe"),
      }, "dir1.1");

      fileSystem.AddDirectory(new List<FSDirectory>
      {
         new FSDirectory("dir1.1.2"),
         new FSDirectory("dir1.1.2")
      }, "dir1.1");


      Console.WriteLine("\n\n FileSystem we created: \n");
      fileSystem.OutputAll();

      FSFile getFileByPathTest = fileSystem.GetFileByPath("dir1/dir1.1/file2.exe");
      FSDirectory getDirectoryByNameInDirectory =
fileSystem.GetFileOrDirectoryInDirectory("dir1.1.2", "dir1.1") as FSDirectory;
```

```
        Console.WriteLine($"\n\nGet file by path test | Trying to get
{@"dir1/dir1.1/file2.exe"}");
        Console.WriteLine($"Received: {getFileByPathTest.id}");

        Console.WriteLine($"\nGet directory by name in directory test | Trying to get
{@"dir1.1.2"} from {@"dir1.1"}");
        Console.WriteLine($"Received: {getDirectoryByNameInDirectory.id}\n\n");

        Console.WriteLine($"Trying to remove {@"dir1.1"} from {@"dir1"}");
        fileSystem.Remove("dir1.1", "dir1");

        Console.WriteLine($"File system after removing {@"dir1.1"} from {@"dir1"}\n\n");
        fileSystem.OutputAll();
    }
}
```

FileSystem.cs

```
#region

using OOP_L2.Utilities;

#endregion

namespace OOP_L2.FilesSystem;

public class FileSystem
{


    public FileSystem(FSDirectory main)
    {
        MainDirectory = main;
    }
    private FSDirectory MainDirectory { get; }


    public FSFile GetFileByPath(string path)
    {
        var pathParts = path.Split('/');
        TreeNode currentDirectory = MainDirectory;
        foreach (var pathPart in pathParts)
        {
            // Console.WriteLine($"Path {pathPart}");

            currentDirectory = currentDirectory.GetChild(pathPart);
        }

        if (currentDirectory is not FSFile || currentDirectory == null)
        {
```

```
            throw new ArgumentException("File not found.");
        }

        return currentDirectory is FSFile ? currentDirectory as FSFile : null;
    }


    public TreeNode GetFileOrDirectoryInDirectory(string id, string from)
    {
        var findFrom = FindDirectory(from);
        if (findFrom.GetChild(id) == null)
        {
            throw new ArgumentException("File or directory nor found.");
        }

        return findFrom.GetChild(id);
    }


    public bool IsDirectory(TreeNode treeNode)
    {
        return treeNode is FSDirectory;
    }


    public void AddFile(List<FSFile> files, string? to = null)
    {
        var directoryToAdd = string.IsNullOrEmpty(to) ? MainDirectory : FindDirectory(to);
        foreach (var file in files)
        {
            try
            {
                directoryToAdd.Add(file);
            }
            catch (Exception e)
            {
                Console.WriteLine($"Error, directory wont be added: {e.Message}");
            }
        }
    }


    public void AddDirectory(List<FSDirectory> directories, string? to = null)
    {
        var directoryToAdd = string.IsNullOrEmpty(to) ? MainDirectory : FindDirectory(to);
        foreach (var directory in directories)
        {
            try
            {
                directoryToAdd.Add(directory);
```

```
            }
            catch (Exception e)
            {
                Console.WriteLine($"Error, directory wont be added: {e.Message}");
            }
        }
    }


    public void Remove(string id, string? from = null)
    {
        FSDirectory directoryToRemoveFrom = string.IsNullOrEmpty(from) ? MainDirectory :
FindDirectory(from);
        directoryToRemoveFrom.RecursiveDescent(node =>
        {
            if (node.id == id)
            {
                if (node.Parent == null)
                {
                    throw new ArgumentException(paramName: nameof(TreeNode), message:
"Cannot remove root directory. ");
                }

                node.Parent.Remove(node.id);
            }
        });
    }


    public void Rename(string file, string renameTo)
    {
    }


    private FSDirectory FindDirectory(string id)
    {
        TreeNode treeNode = null;
        MainDirectory.RecursiveDescent(node =>
        {
            if (node.id == id)
            {
                treeNode = node;
            }
        });

        if (treeNode == null)
        {
            throw new ArgumentException(paramName: nameof(TreeNode), message:
"Directory not found. ");
        }
```

```
        return treeNode is FSDirectory ? treeNode as FSDirectory : MainDirectory;
    }


    private FSFile FindFile(string id)
    {
        TreeNode treeNode = null;
        MainDirectory.RecursiveDescent(_id =>
        {
            if (_id.id == id)
            {
                treeNode = _id;
            }
        });

        if (treeNode == null)
        {
            throw new ArgumentException(paramName: nameof(TreeNode), message:
"Directory not found. ");
        }

        return treeNode is FSFile ? treeNode as FSFile : null;
    }


    public void OutputAll()
    { // output all files and directories from root folder
        MainDirectory.OutputTree("", true);
    }
}

public class FSDirectory : TreeNode
{

    public FSDirectory(string id) : base(id)
    {
        Title = id;
        CreationTime = DateTime.Now;
    }

    public int AttachedFiles
    {
        get
        {
            return _children.Count;
        }
    }

    public DateTime CreationTime { get; }
```

```
    public DateTime LastModificationTime { get; set; }

    public string Title { get; }
}

public class FSFile : TreeNode
{
    private ulong _sizeInBytes;


    public FSFile(string id) : base(id)
    {
        CreationTime = DateTime.Now;
        Title = id;
        _sizeInBytes = (ulong)new Random().Next(0, int.MaxValue);
    }

    public DateTime CreationTime { get; }

    public DateTime LastModificationTime { get; set; }

    public string Title { get; }

    public ulong SizeInBytes
    {
        get
        {
            return _sizeInBytes;
        }
        set
        {
            _sizeInBytes = value;
            LastModificationTime = DateTime.Now;
        }
    }
}
```

TreeNode.cs

```
#region

#endregion

namespace OOP_L2.Utilities;

public class TreeNode
{
    protected readonly Dictionary<string, TreeNode> _children = new Dictionary<string,
TreeNode>();
```

```
    public readonly string id;


    protected TreeNode(string id)
    {
        this.id = id;
    }

    public TreeNode Parent { get; set; }


    public TreeNode GetChild(string id)
    {
        return _children.ContainsKey(id) ? _children[id] : null;
    }


    public void Add(TreeNode item)
    {
        if (_children.ContainsKey(item.id))
        {
            throw new ArgumentException(paramName: nameof(TreeNode), message: "Same
file or directory already exists. ");
        }

        item.Parent?._children.Remove(item.id);

        item.Parent = this;
        _children.Add(item.id, item);
    }


    public void Remove(string id)
    {
        if (!_children.ContainsKey(id))
        {
            throw new ArgumentException(paramName: nameof(TreeNode), message: "There is
no such file or directory.");
        }

        _children.Remove(id);
    }


    public void RecursiveDescent(Action<TreeNode> callback)
    {
        // Console.WriteLine($"Searching for {id} in {this.id}");
        foreach (var childId in _children.Keys)
        {
            // Console.WriteLine($"\t Searching child {childId}");
```

```
            callback(_children[childId]);
            if (_children.ContainsKey(childId))
            {
                _children[childId].RecursiveDescent(callback);
            }
        }
    }


    public void OutputTree(string indent, bool last)
    {
        Console.Write(indent);
        if (last)
        {
            Console.Write("\\-");
            indent += "  ";
        }
        else
        {
            Console.Write("|-");
            indent += "| ";
        }
        Console.WriteLine(id);

        foreach (var id in _children.Keys)
        {
            _children[id].OutputTree(indent, id == _children.Keys.Last());
        }
    }
}
```

Тестування програми

Тестую механізм обробки помилок, додаючи папки та файли із однаковими назвами

```
fileSystem.AddDirectory(directories: new List<FSDirectory>
{
    new FSDirectory(id: "dir1"),
    new FSDirectory(id: "dir1"),
    new FSDirectory(id: "dir2")
});

fileSystem.AddDirectory(directories: new List<FSDirectory>
{
    new FSDirectory(id: "dir1.1"),
    new FSDirectory(id: "dir1.2")
}, to: "dir1");

fileSystem.AddFile(files: new List<FSFile>()
{
    new FSFile(id: "file1.exe"),
    new FSFile(id: "file2.exe"),
    new FSFile(id: "file3.exe"),
}, to: "dir1.1");

fileSystem.AddDirectory(directories: new List<FSDirectory>
{
    new FSDirectory(id: "dir1.1.2"),
    new FSDirectory(id: "dir1.1.2")
}, to: "dir1.1");
```

```
"/Users/vladfrolov/University/4th Semester/OOP/L2/OOP-L2/OOP-L2/OOP-L2/bin/Debug/net7.0/OOP-L2"
Error, directory wont be added: Same file or directory already exists.  (Parameter 'TreeNode')
Error, directory wont be added: Same file or directory already exists.  (Parameter 'TreeNode')
```

Тестування інших реалізованих функцій

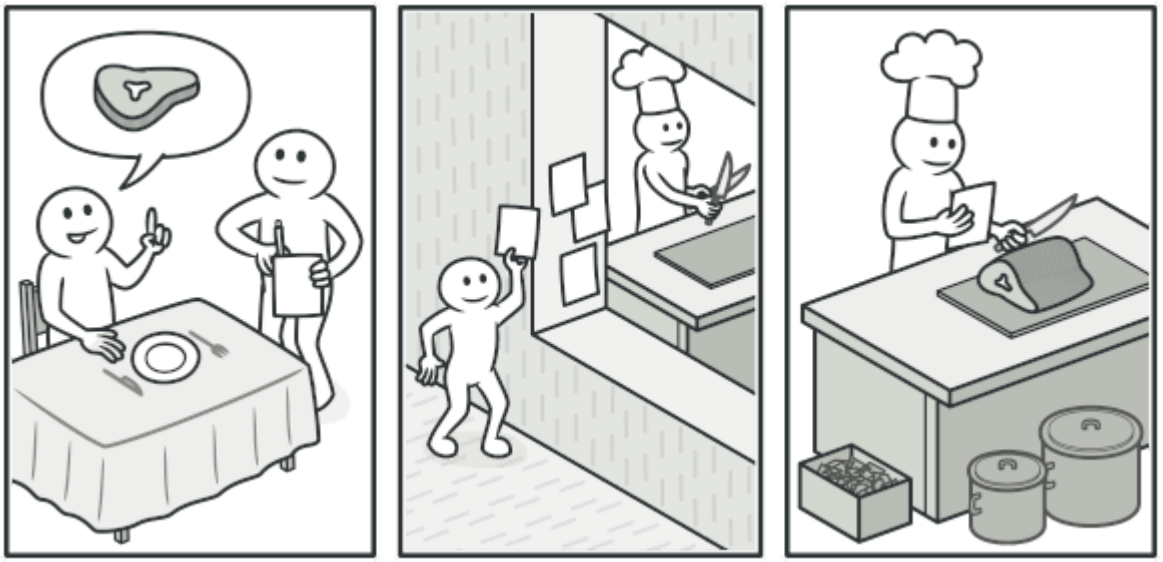
```
Console.WriteLine("\n\n FileSystem we created: \n");
fileSystem.OutputAll();

FSFile getFileByPathTest = fileSystem.GetFileByPath("dir1/dir1.1/file2.exe");
FSDirectory getDirectoryByNameInDirectory = fileSystem.GetFileOrDirectoryInDirectory(id: "dir1.1.2", from: "dir1.1") as FSDirectory;

Console.WriteLine($"\n\nGet file by path test | Trying to get {@"dir1/dir1.1/file2.exe"}");
Console.WriteLine($"Received: {getFileByPathTest.id}");

Console.WriteLine($"\nGet directory by name in directory test | Trying to get {@"dir1.1.2"} from {@"dir1.1"}");
Console.WriteLine($"Received: {getDirectoryByNameInDirectory.id}\n\n");

Console.WriteLine($"Trying to remove {@"dir1.1"} from {@"dir1"}");
fileSystem.Remove(id: "dir1.1", from: "dir1");

Console.WriteLine($"File system after removing {@"dir1.1"} from {@"dir1"}\n\n");
fileSystem.OutputAll();
```

```
 FileSystem we created:

\-root
  |-dir1
  | |-dir1.1
  | | |-file1.exe
  | | |-file2.exe
  | | |-file3.exe
  | | \-dir1.1.2
  | \-dir1.2
  \-dir2


Get file by path test | Trying to get dir1/dir1.1/file2.exe
Received: file2.exe

Get directory by name in directory test | Trying to get dir1.1.2 from dir1.1
Received: dir1.1.2


Trying to remove dir1.1 from dir1
File system after removing dir1.1 from dir1


\-root
  |-dir1
  | \-dir1.2
  \-dir2

Process finished with exit code 0.
```

У даній лабораторній роботі були використані патерни: Command, Iterator(якщо я правильно його реалізував(наче правильно(працює - краще не трогать :D))) та Фасад. А іще реалізована структура TreeNode

Command

В команді нічого поганого не бачу, це базовичок



Використання:

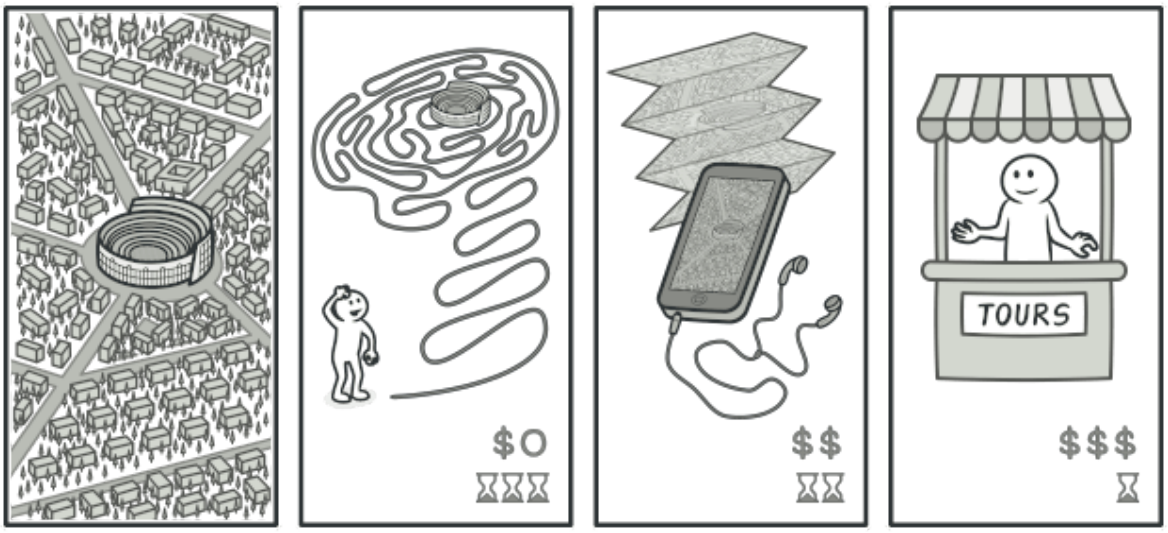
```
public class FSFile : TreeNode
```

```
public class FSDirectory : TreeNode
```

Iterator

Ітератор в даній лабораторній роботі я використовую для рекурсивного спуску по дереву. Від корня до потрібної папки.

Плюси ітератора:
Ітератор дає зручний доступ до всіх елементів колекції, не торкаючись їх реалізації.



Практичне використання у даній лабораторній роботі:
Я вирішив не відділяти ітератор окремим інтерфейсом, а просто запихнув його у класс TreeNode

```
4 usages  Vlad Frolov *
public void RecursiveDescent(Action<TreeNode> callback)
{
    // Console.WriteLine($"Searching for {id} in {this.id}");
    foreach (var childId:string in _children.Keys)
    {
        // Console.WriteLine($"\t Searching child {childId}");
        callback(_children[childId]);
        if (_children.ContainsKey(childId))
        {
            _children[childId].RecursiveDescent(callback);
        }
    }
}
```

Метод RecursiveDescent() рекурсивно проходиться від обраної гілки дерева до її останнього елемента, викликаючи при цьому колбек на всіх етапах проходження

У цьому колбекі знаходиться дія, яку треба виконати для певного елемента. Наприклад пошук папки/файла по id

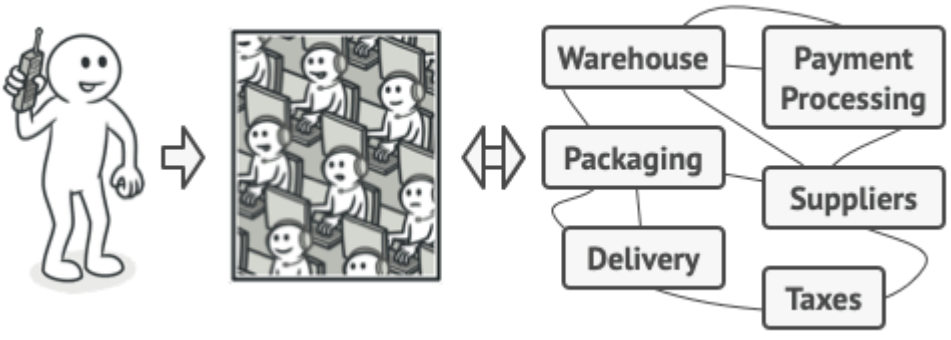
```
TreeNode treeNode = null;
MainDirectory.RecursiveDescent(node =>
{
    if (node.id == id)
    {
        treeNode = node;
    }
});
```

Або реалізація методу Remove()

```
1 usage  new *
public void Remove(string id, string? from = null)
{
    FSDirectory directoryToRemoveFrom = string.IsNullOrEmpty(from) ? MainDirectory : FindDirectory(from);
    directoryToRemoveFrom.RecursiveDescent(node =>
    {
        if (node.id == id)
        {
            if (node.Parent == null)
            {
                throw new ArgumentException(paramName: nameof(TreeNode), message: "Cannot remove root directory. ");
            }

            node.Parent.Remove(node.id);
        }
    });
}
```

Facade

Єдиний мінус - те, що за простим інтерфейсом може бути нечитабельний, складний, запутаний код, в якому найменша зміна зламає правильність виконання задачі інтерфейсу.



Наприклад:

```
fileSystem.AddDirectory( directories: new List<FSDirectory>
{
    new FSDirectory( id: "dir1"),
    new FSDirectory( id: "dir1"),
    new FSDirectory( id: "dir2")
});
```

За AddDirectory() ховається рекурсивний пошук папки, обробка помилкових подій, та саме додавання.