

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій

Кафедра комп'ютерної інженерії та інформаційних систем

Лабораторна робота № 8

з дисципліни «Об'єктно-орієнтовані технології програмування»

на тему:

«Структурні патерни»

Виконав:

студент 1 курсу, групи КІ2м-23-2

_____ Мариняк В.М.

Перевірив:

_____ Лисенко С.М.

Хід роботи

Розглянемо варіант реалізації патернів Адаптер та Декоратор на прикладі інтерфейсу, що надає доступ до потоку даних. За допомогою декораторів буде додано шифрування та стиснення потоку, а адаптер перетворюватиме потік в потрібний формат.

Технічне завдання:

1. Визначити інтерфейс `IDataStream`.
2. Створити конкретний клас `FileStream`, що реалізовуватиме інтерфейс `IDataStream`.
3. Реалізувати абстрактний клас `DataStreamDecorator`, який також реалізує `IDataStream` і слугує базовим класом для конкретних декораторів.
4. Реалізувати конкретні декоратори `EncryptedStream` та `CompressedStream` які відповідно розширюють поточну функціональність.
5. Визначити інтерфейс `IThirdPartyStream`, що виконуватиме роль стороннього потоку даних.
6. Створити конкретний клас `ThirdPartyStream`, який реалізує інтерфейс `IThirdPartyStream`.
7. Реалізувати клас адаптера, який пристосовує потік `IThirdPartyStream` до інтерфейсу `IDataStream`.

Реалізація:

1. Визначення інтерфейсу `IDataStream`

```
1 namespace AdapterAndDecoratorExample.Maryniak.Models;
2
3 public interface IDataStream
4 {
5     void Write(string data);
6     string Read();
7 }
```

2. Створення конкретного класу FileStream

```
1 namespace AdapterAndDecoratorExample.Maryniak.Models;
2
3 public class FileStream : IDataStream
4 {
5     private string data = string.Empty;
6
7     public void Write(string data)
8     {
9         this.data = data;
10        Console.WriteLine($"Writing data to file: {data}");
11    }
12
13    public string Read()
14    {
15        Console.WriteLine($"Reading data from file: {data}");
16        return data;
17    }
18 }
```

3. Визначити абстрактний клас DataStreamDecorator

```
1 namespace AdapterAndDecoratorExample.Maryniak.Models;
2
3 public abstract class DataStreamDecorator(IDataStream dataStream) : IDataStream
4 {
5     public virtual void Write(string data) => dataStream.Write(data);
6
7     public virtual string Read() => dataStream.Read();
8 }
```

4. Реалізація конкретних класів-декораторів

```
1 namespace AdapterAndDecoratorExample.Maryniak.Models;
2
3 public class EncryptedStream(IDataStream dataStream) : DataStreamDecorator(dataStream)
4 {
5     public override void Write(string data)
6     {
7         string encryptedData = Encrypt(data);
8         base.Write(encryptedData);
9     }
10
11    public override string Read()
12    {
13        string encryptedData = base.Read();
14        return Decrypt(encryptedData);
15    }
16
17    private static string Encrypt(string data)
18    => $"ENCRYPTED({data})";
19
20    private static string Decrypt(string encryptedData)
21    => encryptedData.Replace("ENCRYPTED(", "").Replace(")", "");
22 }
```

```

1 namespace AdapterAndDecoratorExample.Maryniak.Models;
2
3 public class CompressedStream(IDataStream dataStream) : DataStreamDecorator(dataStream)
4 {
5     public override void Write(string data)
6     {
7         string compressedData = Compress(data);
8         base.Write(compressedData);
9     }
10
11     public override string Read()
12     {
13         string compressedData = base.Read();
14         return Decompress(compressedData);
15     }
16
17     private static string Compress(string data)
18     => $"COMPRESSED({data})";
19
20     private static string Decompress(string compressedData)
21     => compressedData.Replace("COMPRESSED(", "").Replace(")", "");
22 }

```

5. Визначення інтерфейсу IThirdPartyStream

```

1 namespace AdapterAndDecoratorExample.Maryniak.Models;
2
3 public interface IThirdPartyStream
4 {
5     void SendData(string data);
6     string ReceiveData();
7 }

```

6. Реалізація конкретного класу ThirdPartyStream

```

1 namespace AdapterAndDecoratorExample.Maryniak.Models;
2
3 public class ThirdPartyStream : IThirdPartyStream
4 {
5     private string data = string.Empty;
6
7     public void SendData(string data)
8     {
9         this.data = data;
10        Console.WriteLine($"Sending data to third-party service: {data}");
11    }
12
13    public string ReceiveData()
14    {
15        Console.WriteLine($"Receiving data from third-party service: {data}");
16        return data;
17    }
18 }

```

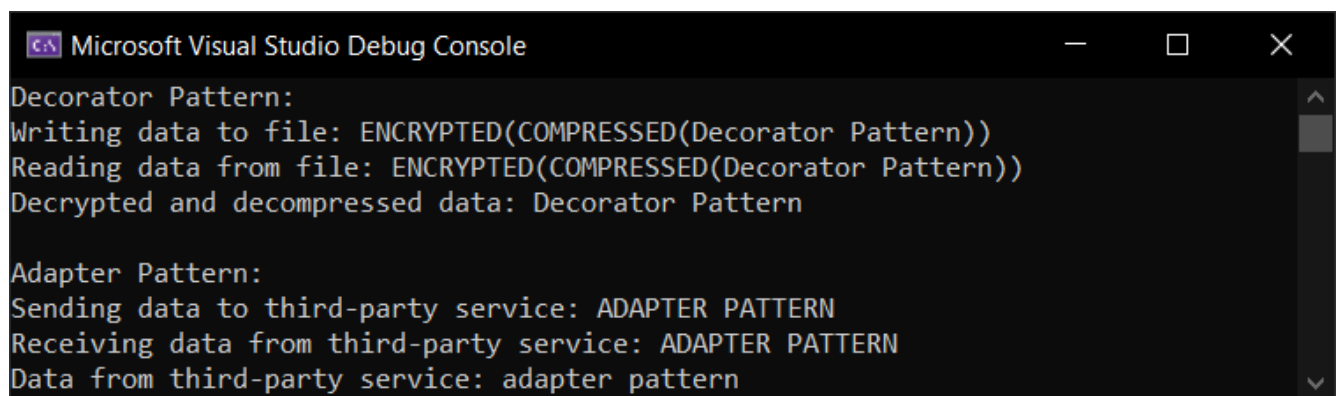
7. Визначення класу адаптера

```
1 namespace AdapterAndDecoratorExample.Maryniak.Models;
2
3 public class ThirdPartyStreamAdapter(IThirdPartyStream thirdPartyStream) : IDataStream
4 {
5     public void Write(string data) => thirdPartyStream.SendData(data.ToUpper());
6
7     public string Read() => thirdPartyStream.ReceiveData().ToLower();
8 }
```

8. Реалізація клієнтського коду

```
1 using AdapterAndDecoratorExample.Maryniak.Models;
2 using FileStream = AdapterAndDecoratorExample.Maryniak.Models.FileStream;
3
4 namespace AdapterAndDecoratorExample.Maryniak;
5
6 internal class Program
7 {
8     static void Main()
9     {
10         Console.WriteLine("Decorator Pattern:");
11
12         IDataStream fileStream = new FileStream();
13         IDataStream encryptedStream = new EncryptedStream(fileStream);
14         IDataStream compressedStream = new CompressedStream(encryptedStream);
15
16         compressedStream.Write("Decorator Pattern");
17         Console.WriteLine($"Decrypted and decompressed data: {compressedStream.Read()}");
18
19         Console.WriteLine("\nAdapter Pattern:");
20
21         IThirdPartyStream thirdPartyService = new ThirdPartyStream();
22         IDataStream thirdPartyAdapter = new ThirdPartyStreamAdapter(thirdPartyService);
23
24         thirdPartyAdapter.Write("Adapter Pattern");
25         Console.WriteLine($"Data from third-party service: {thirdPartyAdapter.Read()}");
26     }
27 }
```

9. Результат виконання програми



```
Microsoft Visual Studio Debug Console
Decorator Pattern:
Writing data to file: ENCRYPTED(COMPRESSED(Decorator Pattern))
Reading data from file: ENCRYPTED(COMPRESSED(Decorator Pattern))
Decrypted and decompressed data: Decorator Pattern

Adapter Pattern:
Sending data to third-party service: ADAPTER PATTERN
Receiving data from third-party service: ADAPTER PATTERN
Data from third-party service: adapter pattern
```

Висновок. Під час лабораторної роботи я навчився реалізовувати патерни Адаптер і Декоратор та створив консольний додаток, який демонструє принципи їхньої роботи.