МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Комп‘ютерний практикум №4

з курсу «Основи розробки програмного забезпечення на платформі Microsoft.NET»

на тему: «Шаблони проектування. Породжуючі шаблони»

Перевірила:
доцент
Ліщук К. І.

Виконав:
студент 2 курсу
групи ІП-21 ФІОТ
Гриценко А. В.

Київ 2024

Варіант 19

Мета: ознайомитися з основними шаблонами проектування, навчитися застосовувати їх при проектуванні і розробці ПЗ.

Постановка задачі комп‘ютерного практикуму № 4
При виконанні комп‘ютерного практикуму необхідно виконати наступні дії:

1) Вивчити породжуючі патерни. Знати загальну характеристику та призначення кожного з них, особливості реалізації кожного з породжуючих патернів та випадки їх застосування.

2) Реалізувати задачу згідно варіанту, запропонованого нижче у вигляді консольного застосування на мові C#. Розробити інтерфейси та класи з застосування одного або декількох патернів. Повністю реалізувати методи, пов‘язані з реалізацією обраного патерну.

3) Повністю описати архітектуру проекту (призначення методів та класів), особливості реалізації обраного патерну. Для кожного патерну необхідно вказати основні класи та їх призначення,

4) Навести UML-діаграму класів

5) Звіт повинен містити:
   a. обґрунтування обраного патерну (чому саме він);
   b. опис архітектури проекту (призначення методів та класів);
   c. UML-діаграму класів
   d. особливості реалізації обраного патерну
   e. текст програмного коду
   f. скріншоти результатів виконання.

Варіант індивідуального завдання:

19) Реалізувати генератор випадкових значень для масивів різних типів даних.

Для реалізації генератора випадкових значень для масивів різних типів даних можна використати породжуючий паттерн "Фабричний метод" (Factory Method). Це патерн, який визначає інтерфейс для створення об'єктів деякого класу, але безпосереднє рішення про те, об'єкт якого класу створювати відбувається в підкласах. Тобто патерн передбачає, що базовий клас делегує створення об'єктів класам спадкоємцям. Оскільки заздалегідь невідомо, об’єкти яких типів необхідно буде створити, то цей патерн для даної задачі є цільовим. На практиці зустрічаються три види патерну «Фабричний метод», у роботі буде реалізовано саме поліморфний фабричний метод, коли визначається інтерфейс фабрики, а за створення конкретного екземпляра продукту відповідає конкретна фабрика.

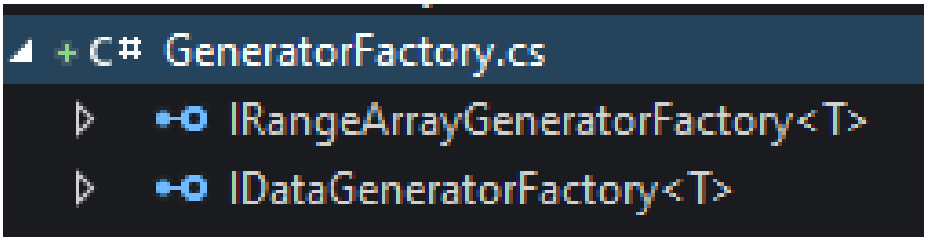

Рисунок 1. IRangeArrayGeneratorFactory<T> та IDataGeneratorFactory<T>

Ці інтерфейси визначають контракти для фабрик, які будуть створювати генератори масивів. IRangeArrayGeneratorFactory<T> використовується для створення генераторів масивів зі значеннями відомого діапазону, а IDataGeneratorFactory<T> - для створення генераторів масивів без вказаного діапазону (наприклад, для генератору булевих значень). Контракт визначається методом Create(). Цей метод має повертати об'єкт, який відповідає певному типу. Таким чином, будь-який клас, який реалізує ці інтерфейси, повинен надати реалізацію методу Create(), яка повертає об'єкт певного типу.

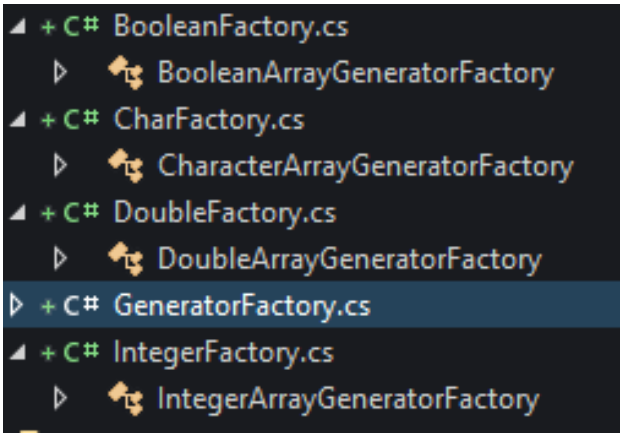

Рисунок 2. Конкретні фабрики

Ці класи реалізують відповідні інтерфейси фабрик і відповідають за створення конкретних екземплярів генераторів масивів.

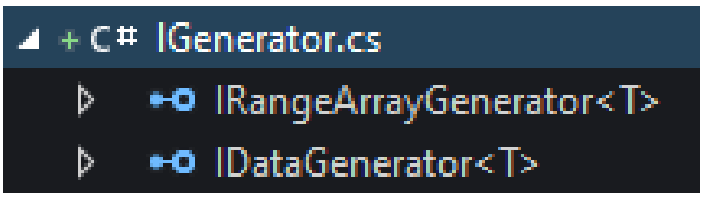

Рисунок 3. IRangeArrayGenerator<T> та IDataGenerator<T>

Ці інтерфейси визначають контракти для генераторів масивів. IRangeArrayGenerator<T> використовується для генерування масивів з відомим діапазоном значень, а IDataGenerator<T> - для генерування масивів з випадковими значеннями без вказаного діапазону

.

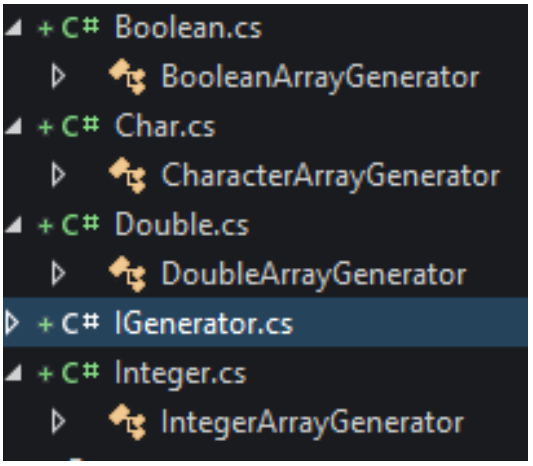

Рисунок 4. Конкретні реалізації генераторів масивів

Ці класи реалізують відповідні інтерфейси генераторів масивів. Кожен з них відповідає за генерацію масиву певного типу даних.

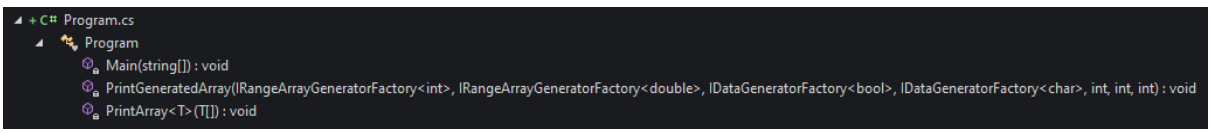

Рисунок 5. Основний клас Program

Дозволяє користувачеві вибирати тип даних для масиву, розмір масиву та відповідно до обраного типу - діапазон значень. Після цього він викликає метод PrintGeneratedArray, щоб вивести на екран згенерований масив. Цей метод відповідає за вибір і виклик відповідного методу генерації масиву залежно від обраного користувачем типу даних. Він також передає відповідні параметри, такі як розмір масиву та діапазон значень, якщо це необхідно. PrintArray метод відповідає за виведення масиву на екран у зручному для користувача форматі.

Особливості реалізації обраного патерну "Фабричний метод"

Гнучкість: реалізація дозволяє легко додавати нові типи генераторів масивів та нові типи даних без необхідності зміни основного коду. Це досягається завдяки використанню інтерфейсів та поліморфізму.

Розділення відповідальностей: rожен клас в вашій реалізації відповідає за конкретну функціональність, що сприяє збереженню коду чистим і організованим.

Інтерфейс визначає загальний інтерфейс об'єктів, які може створювати творець і його підкласи. Конкретні продукти містять код різних продуктів. Продукти будуть відрізнятися реалізацією, але інтерфейс у них буде спільний.

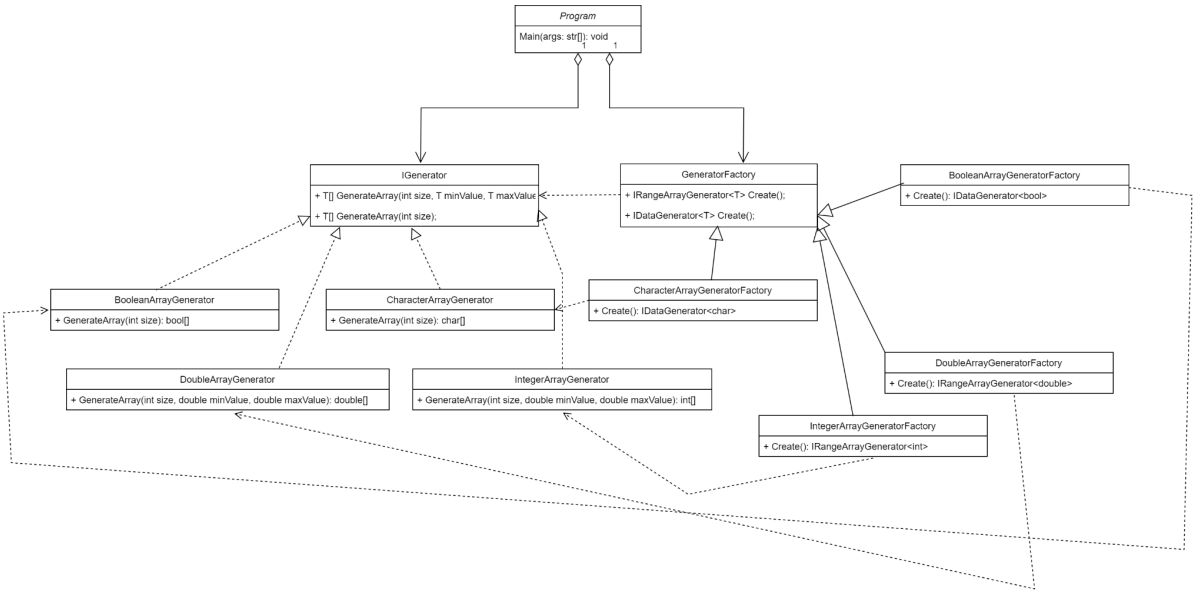

Рисунок 6. UML-діаграма класів

Скріншоті результатів виконання

```
Welcome to the Random Array Generator!

Please select the data type for the array:
1. Integer
2. Double
3. Boolean
4. Character
Enter your choice: 3

Enter the size of the array: 10

Generated array:
[True, False, True, False, True, True, True, True, True, True]

Do you want to repeat the process?
Enter '+' to continue, any other key to exit.
```

```
Welcome to the Random Array Generator!

Please select the data type for the array:
1. Integer
2. Double
3. Boolean
4. Character
Enter your choice: 1

Enter the size of the array: 10

Enter the minimum value for the range: 1
Enter the maximum value for the range: 10

Generated array:
[5, 5, 6, 10, 7, 6, 5, 3, 8, 9]

Do you want to repeat the process?
Enter '+' to continue, any other key to exit.
```

Рисунок 7. Результати роботи програми

Висновок

У даній лабораторній роботі завдання було успішно виконано, обравши патерн Фабричний метод для створення генератора випадкових значень для масивів різних типів даних. Використання цього патерну дозволило реалізувати гнучку систему без прив'язки до конкретних класів, забезпечивши готовність до розширення. Архітектура проекту була структурована, UML-діаграма надала уявлення про його організацію. Особливості реалізації патерну Фабричний метод полягали у делегуванні процесу створення об'єктів підкласам, забезпечуючи гнучкість та розширеність. Код програми було включено до звіту, а скріншоти результатів виконання продемонстрували коректну роботу програми.

Програмний код

BooleanFactory.cs:

```
using laba4.Factories;
using laba4.Realization;

namespace laba4.Factories
{
   public class BooleanArrayGeneratorFactory : IDataGeneratorFactory<bool>
   {
      public IDataGenerator<bool> Create()
      {
         return new BooleanArrayGenerator();
      }
   }
}
```

CharFactory.cs:

```
using laba4.Factories;
using laba4.Realization;

namespace laba4.Factories
{
   public class CharacterArrayGeneratorFactory : IDataGeneratorFactory<char>
   {
      public IDataGenerator<char> Create()
      {
         return new CharacterArrayGenerator();
      }
   }
}
```

DoubleFactory.cs:

```
using laba4.Factories;
using laba4.Realization;

namespace laba4.Factories
{
   public class DoubleArrayGeneratorFactory :
IRangeArrayGeneratorFactory<double>
   {
      public IRangeArrayGenerator<double> Create()
      {
         return new DoubleArrayGenerator();
      }
   }
}
```

IntegerFactory.cs:

```
using laba4.Realization;

namespace laba4.Factories
{
   public class IntegerArrayGeneratorFactory :
IRangeArrayGeneratorFactory<int>
   {
      public IRangeArrayGenerator<int> Create()
      {
         return new IntegerArrayGenerator();
      }
   }
}
```

GeneratorFactory.cs:

```
using laba4.Realization;
```

```
namespace laba4.Factories
{
    public interface IRangeArrayGeneratorFactory<T>
    {
        IRangeArrayGenerator<T> Create();
    }

    // without specifying a range
    public interface IDataGeneratorFactory<T>
    {
        IDataGenerator<T> Create();
    }
}
```

Boolean.cs:

```
using System;

namespace laba4.Realization
{
    public class BooleanArrayGenerator : IDataGenerator<bool>
    {
        public bool[] GenerateArray(int size)
        {
            Random random = new Random();
            bool[] array = new bool[size];
            for (int i = 0; i < size; i++)
            {
                array[i] = random.Next(0, 2) == 1;
            }
            return array;
        }
    }
}
```

Char.cs:

```
using System;

namespace laba4.Realization
{
   public class CharacterArrayGenerator : IDataGenerator<char>
   {
      public char[] GenerateArray(int size)
      {
         Random random = new Random();
         char[] array = new char[size];
         for (int i = 0; i < size; i++)
         {
            // (ASCII value between 65 and 126)
            array[i] = (char)random.Next(65, 122);
         }
         return array;
      }
   }
}
```

Double.cs:

```
using System;

namespace laba4.Realization
{
   public class DoubleArrayGenerator : IRangeArrayGenerator<double>
   {
      public double[] GenerateArray(int size, double minValue, double
maxValue)
      {
         Random random = new Random();
         double[] array = new double[size];
         for (int i = 0; i < size; i++)
         {
            array[i] = minValue + (random.NextDouble() * (maxValue -
minValue));
```

```
            }
            return array;
        }
    }
}
```

Integer.cs:

```
using System;

namespace laba4.Realization
{
    public class IntegerArrayGenerator : IRangeArrayGenerator<int>
    {
        public int[] GenerateArray(int size, int minValue, int maxValue)
        {
            Random random = new Random();
            int[] array = new int[size];
            for (int i = 0; i < size; i++)
            {
                array[i] = random.Next(minValue, maxValue + 1);
            }
            return array;
        }
    }
}
```

IGenerator.cs:

```
namespace laba4.Realization
{
    // with a range parameter
    public interface IRangeArrayGenerator<T>
    {
        T[] GenerateArray(int size, T minValue, T maxValue);
    }
```

```
    // with a single size parameter
    public interface IDataGenerator<T>
    {
        T[] GenerateArray(int size);
    }
}
```

Program.cs:

```
using System;

namespace laba4
{
    class Program
    {
        static void Main(string[] args)
        {
            char repeat;
            do
            {
                Console.WriteLine("\nWelcome to the Random Array Generator!\n");

                Console.WriteLine("Please select the data type for the array:");
                Console.WriteLine("1. Integer");
                Console.WriteLine("2. Double");
                Console.WriteLine("3. Boolean");
                Console.WriteLine("4. Character");

                Console.Write("Enter your choice: ");
                int choice;
                while (!int.TryParse(Console.ReadLine(), out choice) || choice < 1 ||
choice > 4)
                {
                    Console.WriteLine("Invalid choice. Please enter a number between
1 and 4.");
                    Console.Write("Enter your choice: ");
                }
```

```
            Console.Write("\nEnter the size of the array: ");
            int size;
            while (!int.TryParse(Console.ReadLine(), out size) || size <= 0)
            {
                Console.WriteLine("Invalid size. Please enter a positive integer.");
                Console.Write("Enter the size of the array: ");
            }

            int minValue = 0;
            int maxValue = 100;
            if (choice == 1 || choice == 2)
            {
                Console.Write("\nEnter the minimum value for the range: ");
                while (!int.TryParse(Console.ReadLine(), out minValue))
                {
                    Console.WriteLine("Invalid input. Please enter an integer.");
                    Console.Write("Enter the minimum value for the range: ");
                }

                Console.Write("Enter the maximum value for the range: ");
                while (!int.TryParse(Console.ReadLine(), out maxValue))
                {
                    Console.WriteLine("Invalid input. Please enter an integer.");
                    Console.Write("Enter the maximum value for the range: ");
                }
            }

            Factories.IRangeArrayGeneratorFactory<int> intFactory = null;
            Factories.IRangeArrayGeneratorFactory<double> doubleFactory =
null;
            Factories.IDataGeneratorFactory<bool> boolFactory = null;
            Factories.IDataGeneratorFactory<char> charFactory = null;

            switch (choice)
            {
                case 1:
```

```
                intFactory = new Factories.IntegerArrayGeneratorFactory();
                break;
            case 2:
                doubleFactory = new Factories.DoubleArrayGeneratorFactory();
                break;
            case 3:
                boolFactory = new Factories.BooleanArrayGeneratorFactory();
                break;
            case 4:
                charFactory = new Factories.CharacterArrayGeneratorFactory();
                break;
        }

        Console.WriteLine("\nGenerated array:");
        PrintGeneratedArray(intFactory, doubleFactory, boolFactory,
charFactory, size, minValue, maxValue);

        Console.WriteLine("\nDo you want to repeat the process? \nEnter '+'
to continue, any other key to exit.");
        repeat = Console.ReadKey().KeyChar;

        Console.WriteLine();
      } while (repeat == '+');
    }

    static void PrintGeneratedArray(
      Factories.IRangeArrayGeneratorFactory<int> intFactory,
      Factories.IRangeArrayGeneratorFactory<double> doubleFactory,
      Factories.IDataGeneratorFactory<bool> boolFactory,
      Factories.IDataGeneratorFactory<char> charFactory,
      int size, int minValue, int maxValue)
    {
      if (intFactory != null)
      {
        PrintArray(intFactory.Create().GenerateArray(size, minValue,
maxValue));
      }
```

```
            else if (doubleFactory != null)
            {
                PrintArray(doubleFactory.Create().GenerateArray(size, minValue,
maxValue));
            }
            else if (boolFactory != null)
            {
                PrintArray(boolFactory.Create().GenerateArray(size));
            }
            else if (charFactory != null)
            {
                PrintArray(charFactory.Create().GenerateArray(size));
            }
        }

        static void PrintArray<T>(T[] array)
        {
            Console.Write("[");
            for (int i = 0; i < array.Length; i++)
            {
                Console.Write(array[i]);
                if (i < array.Length - 1)
                    Console.Write(", ");
            }
            Console.WriteLine("]");
        }
    }
}
```