**Лабораторная работа №2**

**Полиморфизм**

**Задание:**

Измените, расширьте и опишите иерархию классов из предыдущей лабораторной, используя:

* Описание и наследование классами как минимум трех интерфейсов.
* Абстрактный класс (с абстрактным методом и абстрактным свойством) в качестве основы полиморфизма.

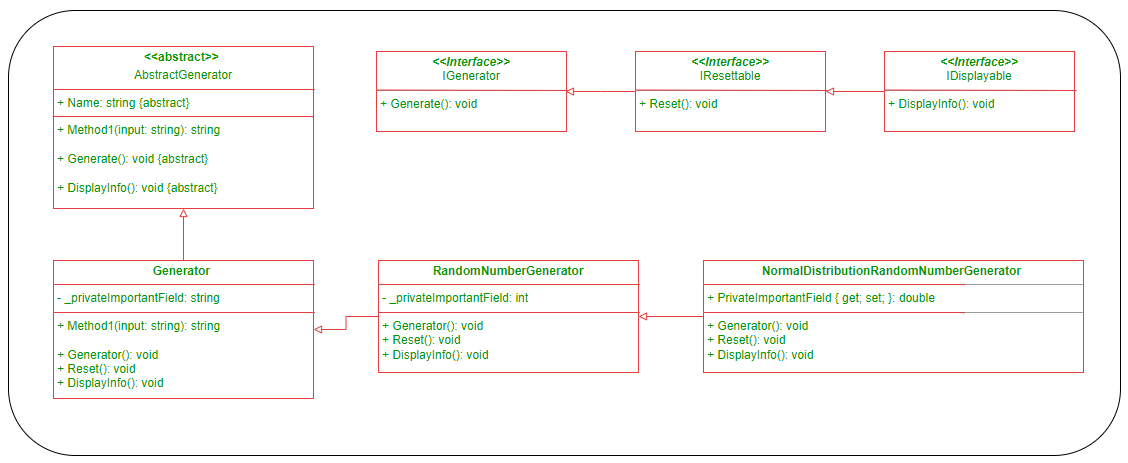
Решение:

Введем абстрактный класс AbstractGenerator, который будет содержать абстрактные методы и свойства.

Реализуем три интерфейса:

* IGenerator для генерации значений;
* IResettable для сброса генератора;
* IDisplayable для отображения информации о генераторе.

Классы Generator, RandomNumberGenerator, и NormalDistributionRandomNumberGenerator будут наследовать от абстрактного класса и реализовывать интерфейсы.



**Абстрактный класс** AbstractGenerator:

Имеет абстрактное свойство Name и метод Generate().

* Метод DisplayInfo() используется для полиморфного отображения информации о генераторе.

**Интерфейсы**:

* IGenerator — для метода Generate(), который должен реализовываться во всех генераторах.
* IResettable — для метода Reset(), который сбрасывает внутренние значения генератора.
* IDisplayable — для метода DisplayInfo(), который отображает информацию о генераторе.

**Классы:**

* **Generator** — базовый класс, реализует абстрактные методы и интерфейсы.
* **RandomNumberGenerator** — наследует от Generator, переопределяет методы для генерации случайных чисел.
* **NormalDistributionRandomNumberGenerator** — наследует от RandomNumberGenerator, добавляет функциональность нормального распределения и переопределяет методы.

В результате мы получаем иерархию классов с применением полиморфизма через абстрактный класс и реализацию интерфейсов для дополнительных возможностей.

Исходный код:

using System;

using System;

namespace lab2

{

// Интерфейс для генерации значений

public interface IGenerator

{

void Generate();

}

// Интерфейс для сброса значений генератора

public interface IResettable

{

void Reset();

}

// Интерфейс для отображения информации

public interface IDisplayable

{

void DisplayInfo();

}

// Абстрактный класс с абстрактным методом и свойством

public abstract class AbstractGenerator

{

public abstract string Name { get; set; }

public abstract void Generate();

public abstract void DisplayInfo();

}

// Класс Generator, наследующий абстрактный класс и реализующий интерфейсы

public class Generator : AbstractGenerator, IGenerator, IResettable, IDisplayable

{

private string \_privateImportantField;

public Generator()

{

\_privateImportantField = "Default Generator";

}

public Generator(string value)

{

\_privateImportantField = value;

}

// Реализация абстрактного свойства

public override string Name { get; set; } = "Base Generator";

// Реализация абстрактного метода

public override void Generate()

{

Console.WriteLine("Base Generator is generating values...");

}

// Реализация метода интерфейса IResettable

public void Reset()

{

\_privateImportantField = "Reset Generator";

}

// Реализация абстрактного метода для отображения информации

public override void DisplayInfo()

{

Console.WriteLine($"Generator: {Name}, Field: {\_privateImportantField}");

}

}

// Класс RandomNumberGenerator, наследующий от Generator

public class RandomNumberGenerator : Generator

{

private int \_privateImportantField;

public RandomNumberGenerator() : base()

{

\_privateImportantField = 42;

Name = "Random Number Generator";

}

public RandomNumberGenerator(int value) : base()

{

\_privateImportantField = value;

}

// Переопределение метода Generate для случайного числа

public override void Generate()

{

Console.WriteLine($"Generating random number: {\_privateImportantField}");

}

public override void DisplayInfo()

{

Console.WriteLine($"RandomNumberGenerator: {Name}, Field: {\_privateImportantField}");

}

// Реализация метода сброса

public new void Reset()

{

\_privateImportantField = 0;

}

}

// Класс NormalDistributionRandomNumberGenerator, наследующий от RandomNumberGenerator

public class NormalDistributionRandomNumberGenerator : RandomNumberGenerator

{

public double PrivateImportantField { get; set; }

public NormalDistributionRandomNumberGenerator() : base()

{

PrivateImportantField = 0.0;

Name = "Normal Distribution Generator";

}

public NormalDistributionRandomNumberGenerator(double value) : base()

{

PrivateImportantField = value;

}

// Переопределение метода Generate для нормального распределения

public override void Generate()

{

Console.WriteLine($"Generating normal distribution value: {PrivateImportantField}");

}

public override void DisplayInfo()

{

Console.WriteLine($"NormalDistributionRandomNumberGenerator: {Name}, Field: {PrivateImportantField}");

}

public new void Reset()

{

PrivateImportantField = 0.0;

}

}

// Основная программа

public class Program

{

public static void Main(string[] args)

{

// Экземпляр Generator

Generator generator = new Generator();

generator.Generate();

generator.DisplayInfo();

generator.Reset();

generator.DisplayInfo();

// Экземпляр RandomNumberGenerator

RandomNumberGenerator rng = new RandomNumberGenerator(100);

rng.Generate();

rng.DisplayInfo();

rng.Reset();

rng.DisplayInfo();

// Экземпляр NormalDistributionRandomNumberGenerator

NormalDistributionRandomNumberGenerator normalRng = new NormalDistributionRandomNumberGenerator(1.5);

normalRng.Generate();

normalRng.DisplayInfo();

normalRng.Reset();

normalRng.DisplayInfo();

}

}

}

Результат:

