МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра информационной безопасности

ОТЧЕТ

по практической работе №3

по дисциплине «Объектно-ориентированное программирование»

Тема: Паттерн “Одиночка”

Студенты гр. 3363 _____ Гончаренко О. Д.
Овсейчик Н. И.,
Минко Д. А.

Преподаватель _____ Новакова Н. Е.

Санкт-Петербург
2024

Цель работы

Изучить и реализовать паттерн проектирования "Одиночка" (Singleton), а также разработать программу с двумя классами:

1. Класс, реализующий последовательный вывод чисел от 0 до 10 с использованием статической переменной;
2. Класс, использующий паттерн "Одиночка" для создания единственного экземпляра.

ХОД РАБОТЫ

1. Изучение паттерна “Одиночка”

Паттерн "Одиночка" (Singleton) гарантирует, что у класса будет только один экземпляр, и предоставляет глобальную точку доступа к этому экземпляру. Это позволяет контролировать доступ к ресурсу или объекту, который должен существовать в единственном числе.

Для реализации паттерна была разработана UML – диаграмма классов, отображающая связи между классами Program, RegularClass, Singleton и SingletonCreator(рис.1).

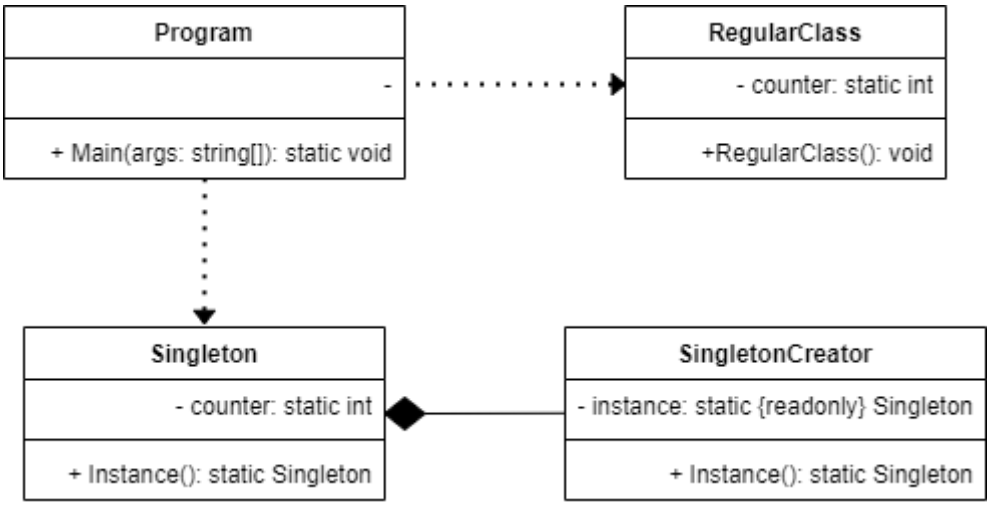

Рисунок 1 – UML диаграмма классов программы

На диаграмме представлены основные классы и их взаимодействие:

- Program управляет созданием и запуском программы. В методе Main создаются экземпляры классов и демонстрируется их работа;
- RegularClass — это класс, использующий статическую переменную counter, которая увеличивается при создании каждого экземпляра;
- Singleton реализует паттерн "Одиночка". Этот класс содержит метод Instance(), который создаёт и возвращает единственный экземпляр объекта;
- SingletonCreator — вспомогательный класс для создания и хранения единственного экземпляра Singleton.

2. Реализация программы

- Создан класс Singleton, реализующий паттерн "Одиночка". Этот класс позволяет создавать только один экземпляр, отслеживая количество обращений через статическую переменную counter.
- Создан класс RegularClass, который использует статическую переменную counter для подсчёта количества созданных экземпляров.
- В классе Program создаются экземпляры классов RegularClass и Singleton, выводятся значения статических переменных, демонстрируется принцип работы паттерна "Одиночка" и отличие от обычного класса.

Спецификация программы

1. Класс Singleton

Описание методов класса Singleton

Таблица 1.1 Описание методов класса Singleton

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
Instance	Singleton	public static	-	Возвращение единственного экземпляра класса Singleton

Описание полей класса Singleton

Таблица 1.2. Описание полей класса Singleton

Имя	Тип	Модификатор доступа	Назначение
counter	int	private static	Подсчёт количества вызовов метода Instance

2. Класс SingletonCreator

Описание методов класса SingletonCreator

Таблица 2.1. Описание методов класса SingletonCreator

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
Instance	Singleton	public static	-	Возвращение единственного экземпляра Singleton, создавая его при первом вызове

Описание полей класса SingletonCreator

Таблица 2.2. Описание полей класса SingletonCreator

Имя	Тип	Модификатор доступа	Назначение
instance	Singleton	private static readonly	Хранение единственного экземпляра Singleton

3. Класс RegularClass

Описание методов класса RegularClass

Таблица 3.1. Описание методов класса RegularClass

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
RegularClass	void	public	-	Конструктор класса, увеличивает значение статического поля counter.

Описание полей класса RegularClass

Таблица 3.2. Описание полей класса RegularClass

Имя	Тип	Модификатор доступа	Назначение
counter	int	private static	Подсчёт количества созданных экземпляров класса RegularClass.

4. Класс Program

Описание методов класса Program

Таблица 4.1. Описание методов класса Program

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
Main	void	public static	string[] args	Точка входа в программу. Создаёт объекты классов, выводит данные, демонстрирует работу Singleton.

Пример работы программы

Программа демонстрирует реализацию паттерна Singleton:

- Создаётся объект Singleton через класс SingletonCreator, обеспечивающий его единственность.
- Также создаётся несколько объектов класса RegularClass для сравнения поведения обычного класса со статическим счётчиком и Singleton.

Методы программы выводят на консоль:

- Количество созданных экземпляров класса RegularClass;
- Общее количество вызовов метода Instance() для получения экземпляра Singleton.

Вывод на консоль демонстрирует работу статического поля counter в обоих классах, показывая, как статические поля хранят данные на уровне класса (рис.2)

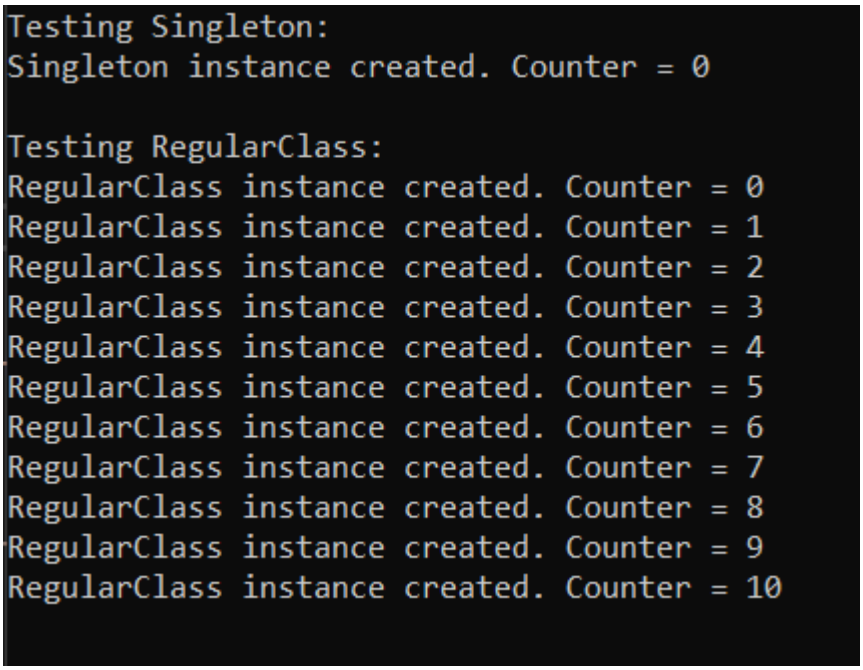

Рисунок 2 – Вывод результата программы

Вывод

В ходе выполнения работы были изучены и реализованы особенности паттерна проектирования "Одиночка" (Singleton), а также проведено сравнение его поведения с обычным классом, использующим статическое поле для подсчёта созданных экземпляров.

1. Паттерн "Одиночка":

- Гарантировал создание единственного экземпляра класса Singleton;
- Обеспечил централизованный доступ к этому экземпляру через статический метод Instance;
- Позволил продемонстрировать, что при многократном вызове Instance создаётся и используется один и тот же объект.

2. Обычный класс:

- Класс RegularClass показал стандартное поведение при создании экземпляров: для каждого вызова создавался новый объект;
- Статическое поле counter продемонстрировало общий счётчик для всех экземпляров класса, который увеличивался при создании каждого объекта;

ПРИЛОЖЕНИЕ 1 ИСХОДНЫЙ КОД ПРОГРАММЫ

Классы Singleton, SingletonCreator

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Pattern3
{
    public class Singleton
    {
        private static int counter = 0;

        // Защищенный конструктор предотвращает создание
экземпляра извне
        protected Singleton()
        {
            Console.WriteLine($"Singleton instance created.
Counter = {counter}");
            counter++;
        }

        private sealed class SingletonCreator
        {
            private static readonly Singleton instance = new
Singleton();
            public static Singleton Instance { get { return
instance; } }
        }

        public static Singleton Instance
        {
            get { return SingletonCreator.Instance; }
        }
    }
}
```

Класс RegularClass

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Pattern3
{
    public class RegularClass
    {
        private static int counter = 0;

        public RegularClass()
        {
            Console.WriteLine($"RegularClass instance created.
Counter = {counter}");
            counter++;
        }
    }
}
```

Класс Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Pattern3;

namespace Pattern3
{
    class Program
    {
        static void Main(string[] args)
        {
            // Проверяем Singleton
            Console.WriteLine("Testing Singleton:");
            Singleton s1 = Singleton.Instance;
            Singleton s2 = Singleton.Instance;

            Console.WriteLine("\nTesting RegularClass:");
            for (int i = 0; i < 11; i++)
            {
                RegularClass instance = new RegularClass();
            }

            Console.ReadKey();
        }
    }
}
```