|  |
| --- |
| Пермский филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский университет  «Высшая школа экономики»  Факультет экономики, менеджмента и бизнес-информатики |
| Образовательная программа бакалавриата «Программная инженерия» |
| **Лабораторная работа № 1**  **«Порождающий паттерн проектирования Одиночка (Singleton)»**   |  |  | | --- | --- | |  | Выполнила:  студентка группы ПИ-15-1  Жигалова Светлана Андреевна | |  | Проверил:  преподаватель кафедры информационных технологий в бизнесе  Кычкин Алексей Владимирович | |

Пермь, 2018 год

Оглавление

[1. Назначение и цель применения паттерна 3](#_Toc506839890)

[2. UML-диаграмма паттерна и пояснение основных элементов 3](#_Toc506839891)

[3. Области применения 3](#_Toc506839892)

[4. Особенности паттерна 3](#_Toc506839893)

[5. Реализация паттерна 4](#_Toc506839894)

[Шаги реализации 4](#_Toc506839895)

[Пример реализации 4](#_Toc506839896)

# Назначение и цель применения паттерна

**Одиночка** — это порождающий паттерн проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

# UML-диаграмма паттерна и пояснение основных элементов



Одиночка определяет статический метод getInstance, который возвращает единственный экземпляр своего класса.

Конструктор одиночки должен быть скрыт от клиентов. Вызов метода getInstance должен быть единственным способом получить объект этого класса.

# Области применения

1. Когда в программе должен быть единственный экземпляр какого-то класса, доступный всем клиентам (например, общий доступ к базе данных из разных частей программы).
2. Одиночка скрывает от клиентов все способы создания нового объекта, кроме специального метода. Этот метод либо создаёт объект, либо отдаёт существующий объект, если он уже был создан.
3. Когда вам хочется иметь больше контроля над глобальными переменными.
4. В отличие от глобальных переменных, Одиночка гарантирует, что никакой другой код не заменит созданный экземпляр класса, поэтому вы всегда уверены в наличие лишь одного объекта-одиночки.

Тем не менее, в любой момент вы можете расширить это ограничение и позволить любое количество объектов-одиночек, поменяв код в одном месте (метод getInstance).

# Особенности паттерна

Одиночка решает сразу две проблемы (нарушая *принцип единственной ответственности* класса).

1. Гарантирует наличие единственного экземпляра класса. Чаще всего это полезно для доступа к какому-то общему ресурсу, например, базе данных.
2. **Предоставляет глобальную точку доступа.** Это не просто глобальная переменная, через которую можно достучаться к определённому объекту. Глобальные переменные не защищены от записи, поэтому любой код может подменять их значения без вашего ведома.

|  |  |
| --- | --- |
| **Преимущества** | **Недостатки** |
| Гарантирует наличие единственного экземпляра класса. | Нарушает *принцип единственной ответственности класса*. |
| Предоставляет к нему глобальную точку доступа. | Маскирует плохой дизайн. |
| Реализует отложенную инициализацию объекта-одиночки. | Проблемы мультипоточности. |
| Требует постоянного создания Mock-объектов при юнит-тестирования. |

# Реализация паттерна

## Шаги реализации

1. Добавьте в класс приватное статическое поле, которое будет содержать одиночный объект.
2. Объявите статический создающий метод, который будет использоваться для получения одиночки.
3. Добавьте «ленивую инициализацию» (создание объекта при первом вызове метода) в создающий метод одиночки.
4. Сделайте конструктор класса приватным.
5. В клиентском коде замените вызовы конструктора вызовами создающего метода.

## Пример реализации

Классическая реализация данного шаблона проектирования на C# выглядит следующим образом:

class Singleton

{

    private static Singleton instance;

    private Singleton()

    {}

    public static Singleton getInstance()

    {

        if (instance == null)

            instance = new Singleton();

        return instance;

    }

}

Конкретный пример реализации.

Пусть на каждом компьютере можно одномоментно запустить только одну операционную систему. В этом плане операционная система будет реализоваться через паттерн синглтон:

class Program

{

    static void Main(string[] args)

    {

        Computer comp = new Computer();

        comp.Launch("Windows 8.1");

        Console.WriteLine(comp.OS.Name);

        // у нас не получится изменить ОС, так как объект уже создан

        comp.OS = OS.getInstance("Windows 10");

        Console.WriteLine(comp.OS.Name);

        Console.ReadLine();

    }

}

class Computer

{

    public OS OS { get; set; }

    public void Launch(string osName)

    {

        OS = OS.getInstance(osName);

    }

}

class OS

{

    private static OS instance;

    public string Name { get; private set; }

    protected OS(string name)

    {

        this.Name=name;

    }

    public static OS getInstance(string name)

    {

        if (instance == null)

            instance = new OS(name);

        return instance;

    }

}