**Паттерн “Одиночка” (Singleton)**

**Что делает?**

Гарантирует, что у класса есть только один экземпляр, и предоставляет к нему

глобальную точку доступа.

**Какие проблемы решает?**

Паттерн решает такие проблемы как:

1. Как можно гарантировать, что у класса есть только один экземпляр?
2. Как можно легко получить доступ к единственному экземпляру класса?
3. Как класс может контролировать свой экземпляр?
4. Как имитировать глобальную переменную?

**Примеры**

1. Рассмотрим консольную игру “Крестики-нолики”. Хотим иметь класс Game, в котором будет находится вся основная логика. Понятно, что в рамках одной программы должен находится только один экземпляр этого класса, поэтому Game может использовать паттерн “Одиночка”. Также в игре есть класс Field, в котором будет хранится игровое поле, и несколько методов для взаимодействия с ним. Опять же понятно, что в игре должно быть только одно поле, поэтому при реализации класса Field также стоит использовать этот шаблон.
2. Конфигурационный файл. С помощью singleton мы можем вынести все конфигурационные переменные в один файл, таким образом мы не будет загрязнять глобальное пространство имен лишними переменными, а также обеспечим одновременный доступ к файлу.
3. Похожим образом можем использовать singleton и для генерации журнального файла (log file)

**Пример из жизни**

Правительство государства — хороший пример одиночки. В государстве может быть только одно официальное правительство. Вне зависимости от того, кто конкретно заседает в правительстве, оно имеет глобальную точку доступа «Правительство страны N».

**Как реализовать?**

1. Глобальная переменная дает доступ к объекту, но не запрещает инстанцировать класс в нескольких экземплярах.
2. Более удачное решение - сам класс контролирует то, что у него есть только один экземпляр, может запретить создание дополнительных экземпляров, перехватывая запросы на создание новых объектов, и он же способен предоставить доступ к своему экземпляру.

**Структура**

|  |
| --- |
| Singleton |
| - instance : Singleton  ... |
| - Singleton()  + getInstance() : Singleton  ... |

**Реализация на Java**

// File Name: Singleton.java

public class Singleton {

private static Singleton singleton = new Singleton( );

/\* A private Constructor prevents any other

\* class from instantiating.

\*/

private Singleton() { }

/\* Static 'instance' method \*/

public static Singleton getInstance( ) {

return singleton;

}

}

// File Name: SingletonDemo.java

public class SingletonDemo {

public static void main(String[] args) {

Singleton tmp = Singleton.getInstance();

}

}

По существу переменные экземпляров, объявленные как static, являются глобальными переменными. При объявлении объектов их класса программа не создаёт никаких копий статической переменной. Вместо этого все экземпляры класса совместно используют одну и ту же статическую переменную.

**Реализация на Kotlin**

object в Kotlin реализует шаблон “Одиночка”

**object** Singleton {}

**Небольшой пример**

// File Name: Singleton.kt

**object** Singleton {

**var number** = 0

}

// File Name: main.kt

**fun** main() {

**val** instance1 = Singleton

*println*(instance1.**number**) *// 0*

**val** instance2 = Singleton

instance1.**number** = 10

*println*(instance2.**number**) *// 10*

}

**Отношения с другими паттернами**

* Фасад можно сделать Одиночкой, так как обычно нужен только один объект-фасад.
* Паттерн Легковес может напоминать Одиночку, если для конкретной задачи его получилось свести количество объектов к одному. Но между паттернами есть два кардинальных отличия:
  + В отличие от Одиночки, можно иметь множество объектов-легковесов.
  + Объекты-легковесы должны быть неизменяемыми, тогда как объект-одиночка допускает изменение своего состояния.
* Абстрактная фабрика, Строитель и Прототип могут быть реализованы при помощи Одиночки.

**Вывод**

1. Паттерн следует использовать, если мы хотим иметь ровно один экземпляр некоторого класса, легко доступный всем клиентам
2. Синглтоны предпочтительнее глобальных переменных, потому что
   1. Они не загрязняют глобальное пространство имен ненужными переменными.
   2. Они допускают отложенное выделение памяти и инициализацию, тогда как глобальные переменные во многих языках всегда потребляют ресурсы.
3. С помощью паттерна одиночка могут быть реализованы многие другие паттерны