12. Шаблон Сек (Singleton)

ЛЕКЦИОНЕН КУРС: ШАБЛОНИ ЗА ПРОЕКТИРАНЕ

ДОЦ. ЕМИЛ ДОЙЧЕВ

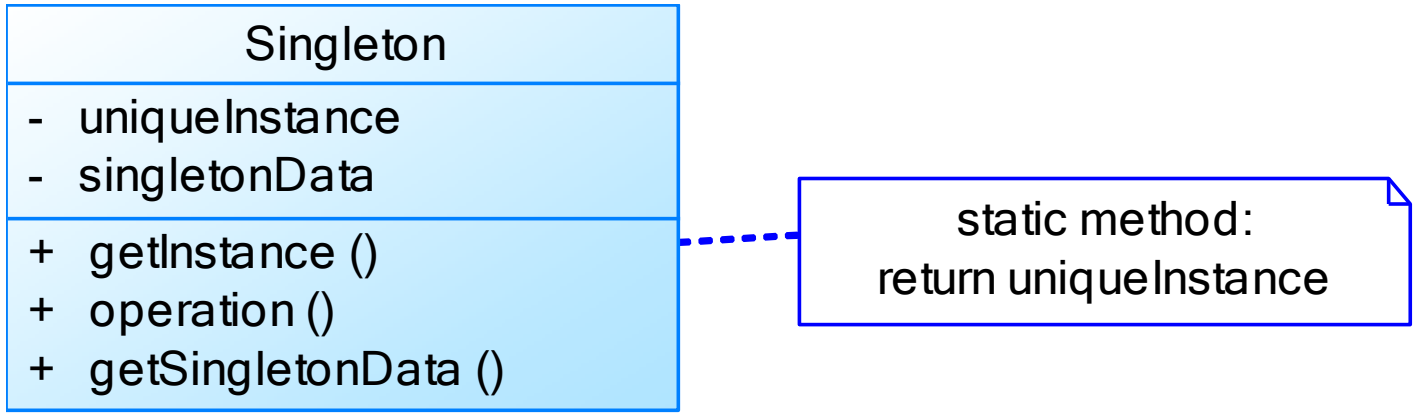
Общи сведения

- ✓ **Вид:** Създаващ за обекти
- ✓ **Цел:** Гарантира, че даден клас може да има само една инстанция и предоставя глобален достъп до нея.
- ✓ **Известен и като:** няма алтернативни наименования

Мотивация и приложимост

- ✓ Понякога е необходимо даден клас да има само една инстанция.
- ✓ Например – необходим ни е само един мениджър на прозорци в десктоп приложение. Или само една „фабрика“ за семейство от обекти.
- ✓ Тази единствена инстанция трябва да е лесно достъпна.
- ✓ Трябва да гарантираме, че допълнителни инстанции от този клас не могат да бъдат създавани.

Структура


Singleton
- uniqueInstance
- singletonData
+ getInstance ()
+ operation ()
+ getSingletonData ()
static method:
return uniqueInstance

Следствия

- ✓ **Предимства**
  - Контролиран достъп до единствената инстанция

Имплементация

- ✓ Използваме статичен метод за да дадем възможност на клиентите да получат достъп до единствената инстанция.
  - Защо?
- ✓ Използваме private конструктор.
  - Защо?

```
public class SingletonV1 {

    // Референция към единствената инстанция
    private static SingletonV1 uniqueInstance = null;

    // Някакъв атрибут на инстанцията.
    private int data = 0;

    /**
     * Връща референция към единствената инстанция. Създава инстанцията ако все
     * още не съществува (това се нарича lazy (мързелива, отложена)
     * инициализация)
     */
    public static SingletonV1 getInstance() {
        if (uniqueInstance == null)
            uniqueInstance = new SingletonV1();

        return uniqueInstance;
    }

    /**
     * Singleton конструктора. Забележете, че е private! Външен клас не може да
     * създаде инстанция.
     */
    private SingletonV1() {

    }

    public int getData() {
        return data;
    }

    public void setData(int data) {
        this.data = data;
    }
}
```

Имплементация

- ✓ Singleton инстанцията се създава само ако е необходима. Това се нарича *lazy instantiation* (отложено, мързеливо създаване на обект).
- ✓ Какво ще се случи ако две нишки едновременно извикат getInstance()? Възможно ли е да възникне проблем?
  - Възможно е да бъдат създадени две инстанции на класа Singleton.
  - Как можем да предотвратим това?
    - Като направим метода getInstance() синхронизиран.

Имплементация

- ✓ Имплементация със синхронизация.
  - Thread safe
  - Синхронизацията е скъпа операция и освен това е необходима само докато се създаде първата инстанция.
  - Решение: създаване на предварителна инстанция (eager instantiation) вместо отложена.

```
public class SingletonV2 {

    // Референция към единствената инстанция.
    private static SingletonV2 uniqueInstance = null;

    // Някакъв атрибут на инстанцията.
    private int data = 0;

    /**
     * Връща референция към единствената инстанция. Създава инстанцията ако все
     * още не съществува (това се нарича lazy (мързелива, отложена)
     * инициализация). Част от тялото на метода е синхронизирано за да е Thread Safe.
     */
    public static SingletonV2 getInstance() {
        synchronized (SingletonV2.class) {
            if (uniqueInstance == null)
                uniqueInstance = new SingletonV2();
        }

        return uniqueInstance;
    }

    /**
     * Singleton конструктора. Забележете, че е private! Външен клас не може да
     * създаде инстанция.
     */
    private SingletonV2() {

    }

    public int getData() {
        return data;
    }

    public void setData(int data) {
        this.data = data;
    }
}
```

Имплементация

- ✓ Имплементация с предварителна (eager) инициализация.
- ✓ Гарантирано thread safe!

```
public class SingletonV3 {

    // Референция към единствената инстанция с предварително (eager) създаване.
    private static SingletonV3 uniqueInstance = new SingletonV3();

    // Някакъв атрибут на инстанцията.
    private int data = 0;

    /**
     * Връща референция към единствената инстанция.
     */
    public static SingletonV3 getInstance() {
        return uniqueInstance;
    }

    /**
     * Singleton конструктора. Забележете, че е private! Външен клас не може да
     * създаде инстанция.
     */
    private SingletonV3() {

    }

    public int getData() {
        return data;
    }

    public void setData(int data) {
        this.data = data;
    }
}
```

Singleton с наследяване

- ✓ Какво ще се случи ако искаме да наследим Singleton клас и по този начин единствената инстанцията да е инстанция на класа наследник?
- ✓ Например: FooFactory има наследници BarFactory и AgentFactory. Искаме да инстанцираме само един factory – BarFactory или AgentFactory.
- ✓ Как можем да го реализираме?
  - Правим така, че статичния getInstance() метод на FooFactory да определи кой точно клас трябва да инстанцира. Това може да се направи чрез параметър на метода или с променлива от обкръжението (environment variable).
    - В този случай конструкторите на подкласовете не могат да бъдат private (защо?) и клиентите *могат* да създадат допълнителни инстанции.
  - Всеки подклас предоставя getInstance() метод.
    - По този начин конструкторите на подкласовете могат да бъдат private.

Имплементация

- ✓ Определяне на класа за инстанциране, чрез параметър на getInstance() метода.
- ✓ Използване:
  - `FooFactory factory = FooFactory.getInstance("agent");`
  - `FooFactory factory = FooFactory.getInstance();`

```
public class FooFactory {

    // Референция към единствената инстанция
    private static FooFactory uniqueInstance = null;

    /**
     * Връща референция към единствената инстанция. Ако инстанцията все още не
     * съществува създава "Bar" по подразбиране.
     */
    public static FooFactory getInstance() {
        if (uniqueInstance == null)
            uniqueInstance = new BarFactory();

        return uniqueInstance;
    }

    /**
     * Връща референция към единствената инстанция. Ако инстанцията все още не
     * съществува определя според параметъра кой клас да инстанцира.
     * Има ли проблем тук?
     */
    public static FooFactory getInstance(String name) {
        if (uniqueInstance == null) {
            if (name.equals("bar"))
                uniqueInstance = new BarFactory();
            else if (name.equals("agent"))
                uniqueInstance = new AgentFactory();
        }

        return uniqueInstance;
    }

    /**
     * Констурктора не може да бъде private тук!
     */
    protected FooFactory() {

    }
}
```

Имплементация

- ✓ Текущата имплементация на getInstance() нарушава open-closed принципа: трябва да се модифицира при добавяне на нов подклас на FooFactory.
- ✓ Това може да се избегне ако като параметър се предава името на класа, който трябва да бъде инстанциран:

```
public static FooFactory getInstance(String name)
        throws InstantiationException, IllegalAccessException, ClassNotFoundException {
    if (uniqueInstance == null)
        uniqueInstance = (FooFactory)Class.forName(name).newInstance();

    return uniqueInstance;
}
```

Имплементация

- ✓ Реализация чрез getInstance() метод във всеки клас наследник.
- ✓ Конструкторите са private.
- ✓ Използване:
  - `FooFactoryV2 factory = BarFactoryV2.getInstance();`
  - `FooFactoryV2 factory = FooFactoryV2.getInstance();`
    - ✓ Може да върне null ако не е създаден инстанция.

```
public abstract class FooFactoryV2 {

    // Референция към единствената инстанция
    protected static FooFactoryV2 uniqueInstance = null;

    /**
     * Връща референция към единствената инстанция.
     */
    public static FooFactoryV2 getInstance() {
        return uniqueInstance;
    }

    /**
     * Конструктора не може да бъде private тук!
     */
    protected FooFactoryV2() {

    }
}
```

```
public class BarFactoryV2 extends FooFactoryV2 {
    /**
     * Връща референция към единствената инстанция.
     */
    public static FooFactoryV2 getInstance() {
        if (uniqueInstance == null)
            uniqueInstance = new BarFactoryV2();

        return uniqueInstance;
    }

    /**
     * Конструктора е private тук!
     */
    private BarFactoryV2() {

    }
}
```

Край: Шаблон Сек

ЛЕКЦИОНЕН КУРС: ШАБЛОНИ ЗА ПРОЕКТИРАНЕ