singleton

---

tags: #паттерны
prev:: 🔗 назад в библиотеку
category::
url::
children::

автоматически порождпется единственный экземпляр класса
другие экземпляры класса порождать невозможно
порожденный экземпляр класса доступен всем и всегда

> private static final ... INSTANCE;
> private constructor (тут могут быть константы, например имена файлов и тд = настройки)
> + публичный метод получения экземпляра - getter (return INSTANCE)

Использование

- поздняя или потокобезопасная инициализация поля INSTANCE
- перепорождение экзмепляра (при изменении настроек или сброса значений полей)
  - порождение нескольких экземпляров, но в заранее известном числе (для баланса нагрузки, например булевская проверка занятости состояния *(чекать например загрузку экрана в стейте, проверять во вью? чекнуть инфу)*, или в константу можно написать кол-во повторений-счетчик)
- порождение нескольких экземпляров с разным функционалом (= заменяется в джаве enum`ы)

нарушает принцип 🔗 single responsibility, тк еще и создает свой экземпляр сам = уникальность (ну и + выполняет свои функции = функциональность)

- [ ] **написать про модифайер в андроиде - экземпляр должен расширяться путем порождания подклассов , и клиетам нужно иметь возможность работать с расширенным экземпляром без модификации своего кода**
- [ ] **написать что это на доп баллы**

🔗 статистический класс или синглтон?

```java
package ask.urfu.examples.patterns.creation.singleton;

/**
 * Singleton pattern
 */
public class LonelyRanger {

  /**
   * The one and only instance
   */
  private static final LonelyRanger INSTANCE = new LonelyRanger();

  //insert any needed fields here

  /**
   * Private constructor ensures no other instance ever appears
   */
  private LonelyRanger() {
    //initialize any needed fields here
  }

  /**
   * Everyone can call the one and only instance this way
   *
   * @return the one and only instance
   */
  public static LonelyRanger instance() {
    return INSTANCE;
  }

  /**
   * Functionality
   */
  public void saveEverybody() {
    //insert code here
  }
}
```

```java
package ask.urfu.examples.patterns.creation.singleton;
```

```
/**
 * Weird kind of singletons -- enum constants
 */
public enum Superhero {

  BATMAN {
    //insert fields here

    @Override
    public void saveTheWorld() {
      //insert code here
    }
  },

  SUPERMAN {
    //insert fields here

    @Override
    public void saveTheWorld() {
      //insert code here
    }
  };

  //insert fields and constructor here

  public abstract void saveTheWorld();
}
```

использование:

```
package ask.urfu.examples.patterns.creation.singleton;

/**
 * How to use Singleton pattern
 */
public class Usage {

  public static void main(String[] args) {

    //this is how we use a singleton -- we call it, use its functionality, and
feel happy
    LonelyRanger.instance().saveEverybody();

    //weird singletons
```

```
    Superhero.BATMAN.saveTheWorld();
    Superhero.SUPERMAN.saveTheWorld();
  }

}
```

1. **Enum** `Superhero` имеет только два возможных значения: `BATMAN` и `SUPERMAN`. Это гарантирует, что в JVM будет только один экземпляр каждого супергероя.
2. Каждое значение `enum` является статическим и финальным, что означает, что они неизменяемы и доступны без создания экземпляра класса.
3. Использование `enum` автоматически обеспечивает потокобезопасность и избавляет от необходимости реализовывать методы `getInstance()` или `INSTANCE`.
4. Каждое значение `enum` может иметь свои собственные поля и методы, как показано для методов `saveTheWorld()`.
5. Создание экземпляров супергероев происходит автоматически при загрузке класса и не требует дополнительных усилий со стороны программиста.

🦁 Клинт Иствуд

---

prev:: 🔗 назад в библиотеку