OBJEKTUMVEZÉRTELT RENDSZEREK TERVEZÉSE

4. gyakorlat

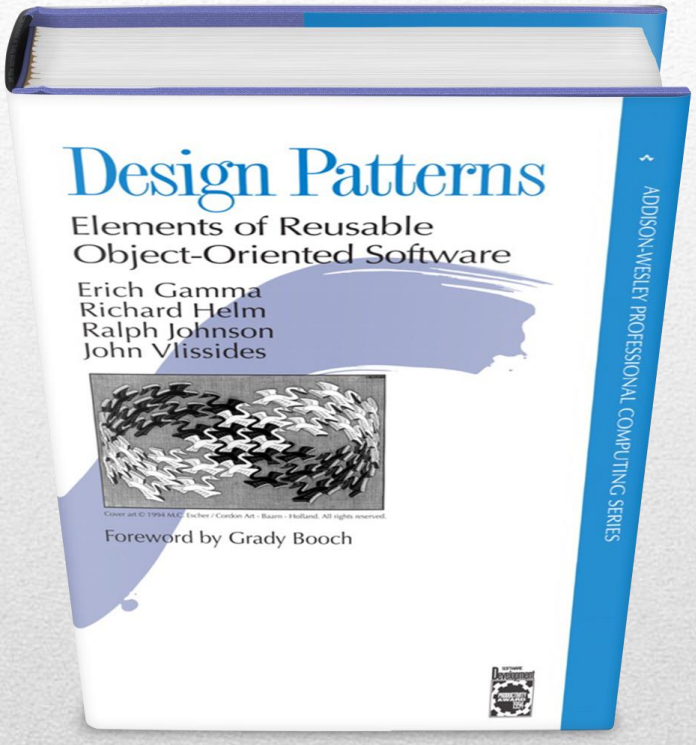
- https://classroom.github.com/g/5_Fe5VCP

Github

TERVEZÉSI MINTÁK

Bevezető

Inspiráció



- Könyv: Design Patterns: Elements of Reusable object-Oriantated Software
- Minta leírások és katalógus
- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
- → „Gang of Four” (GOF)

Szoftvertervezés folyamata

Követelmények megvizsgálása

Problémák azonosítása

Megoldás megtervezése

Ismétlődő feladat

Időigényes

Minden tervezési minta **leír** egy gyakran előforduló **problémát** és **megadja** a hozzá tartozó **megoldás** lényegét.

Megoldás: tervezési minták használata

Előnyök

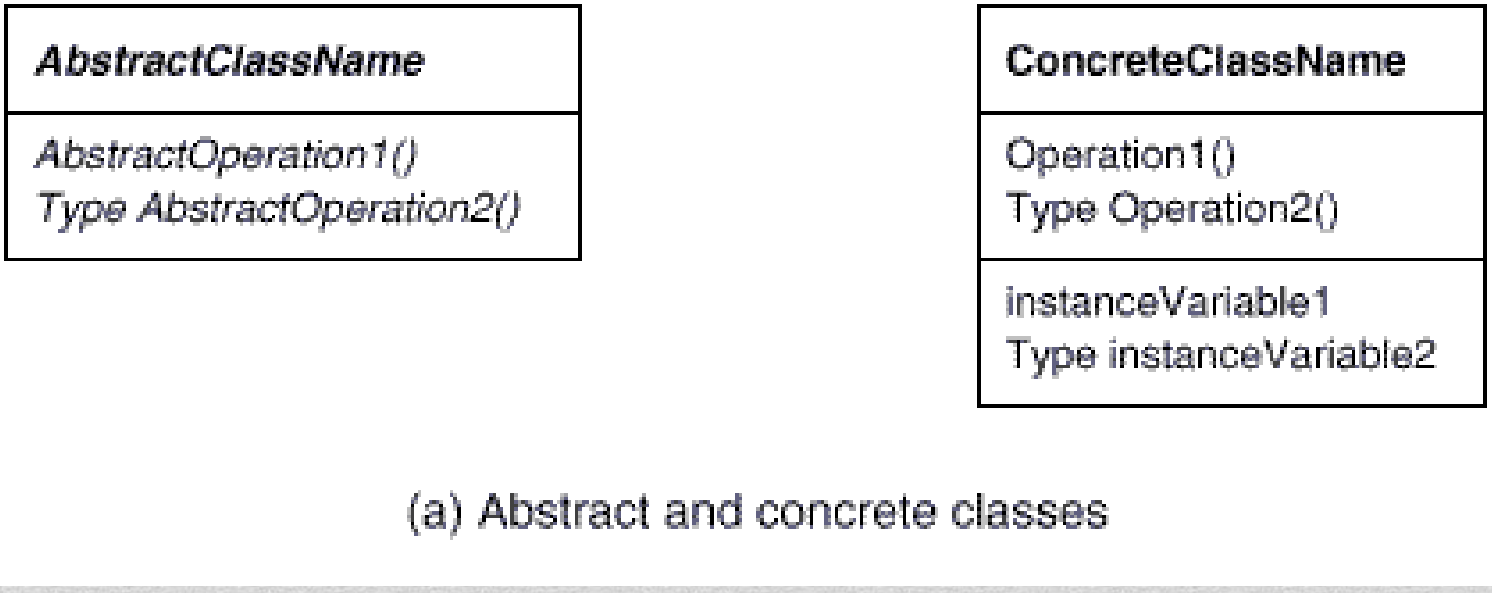
- Jobb szoftver dizájn
- Javuló csapat kommunikáció
  - Közös koncepciók alakulnak ki
- Hatékonyabb probléma megoldás
- Példák a Java beépített függvénykönyvtárában:
  - https://stackoverflow.com/a/2707195

Tervezési minták felépítése

- **Minta neve**
  - Kommunikáció, közös nyelv
  - Absztrakt szinten való tervezést tesz lehetővé
- **Probléma leírása**
  - Hol lehet alkalmazni. Kontextust írja le.
  - Megadja mi a probléma a mostani módszerrel.
- **Problémára nyújtott megoldás**
  - Megadja a megoldás sablonját.
  - Leírja a megoldásban szereplő elemek szerepét.
- **Következmény**
  - Kompromisszumok, alternatívák, költségek és nyereségek

Minták jelölése: OMT notations

- OMT (Object Modeling Technique)
- Nem UML! (de azért nagyon hasonló)
- Gamma et al. használják minták megadásához
- Osztály, Object, Interaction diagramok-at ír le

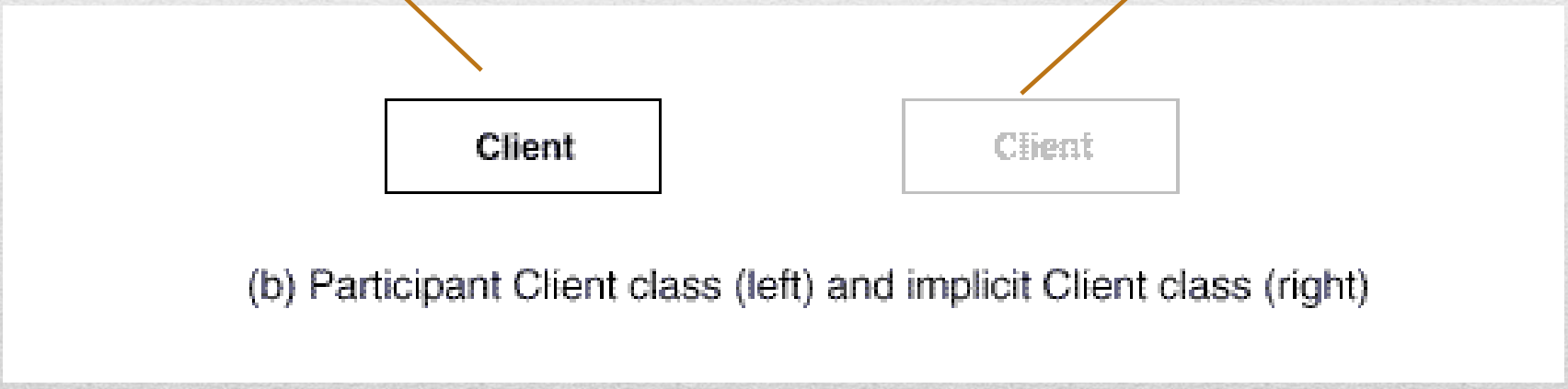


(a) Abstract and concrete classes

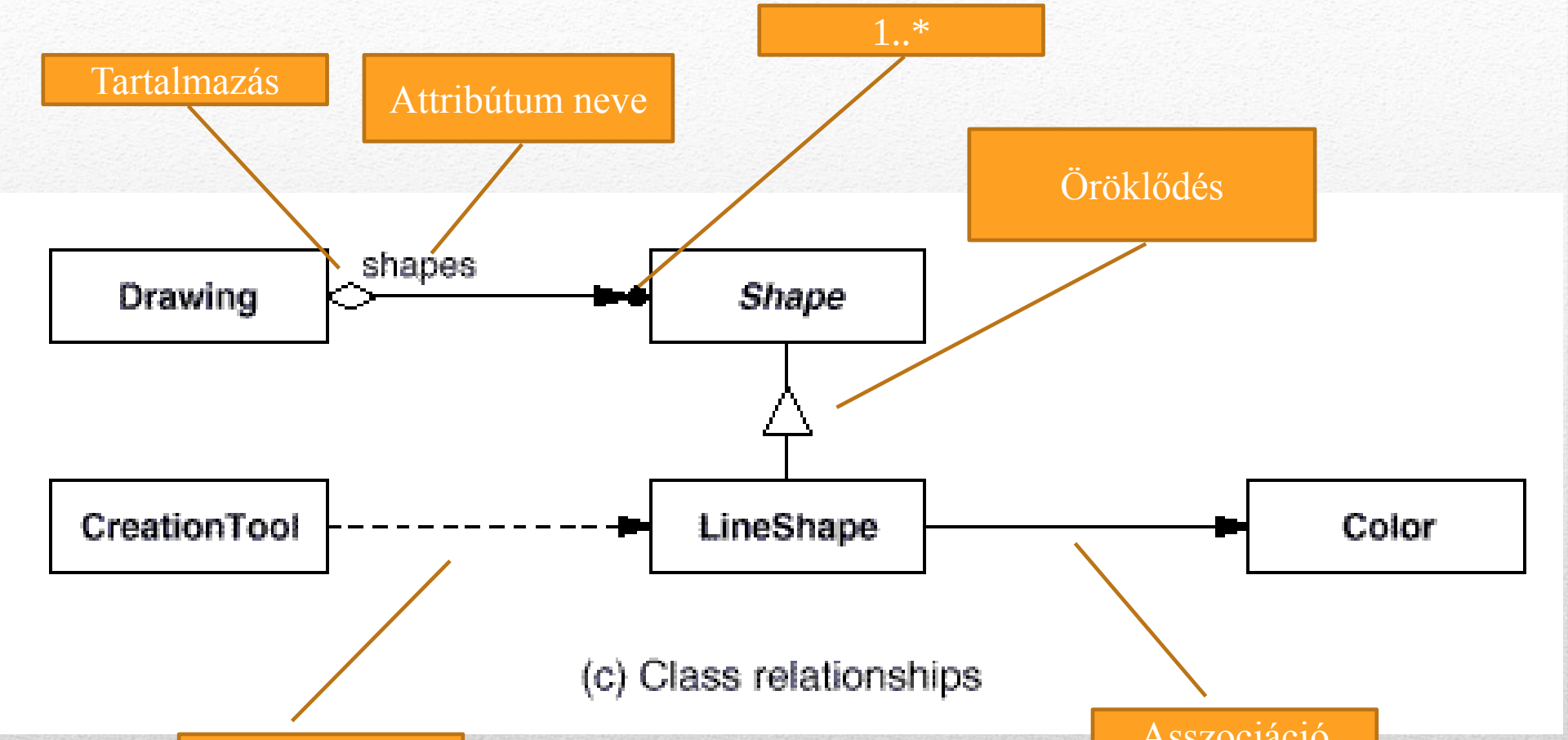
OMT notations

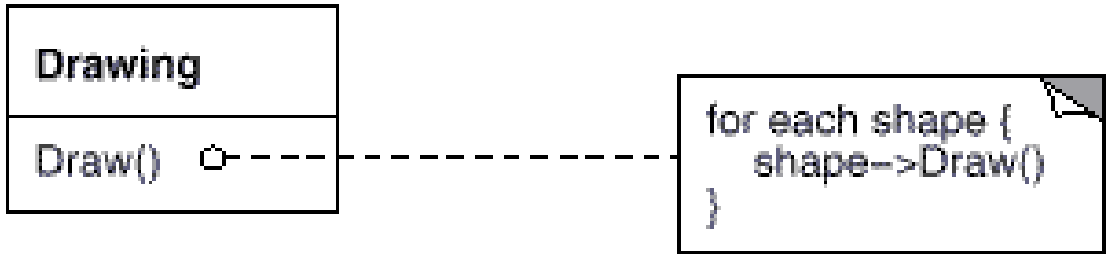
A kliensnek tényleges szerepe van a mintában.

A kliensnek szerepe nincsen meghatározva a mintában, de a megértésben segíthet melyik osztályokkal állhat kapcsolatban.



(b) Participant Client class (left) and implicit Client class (right)


Tartalmazás
Attribútum neve
1..*
Öröklődés
Drawing
shapes
Shape
CreationTool
LineShape
Color
(c) Class relationships
példányosítás
Asszociáció



(d) Pseudocode annotation

OMT notations

Tervezési minta katalógus

- Sok tervezési mintát tartalmaz
- Segít a minták megtalálásában

Szerkezeti	Gyártási	Viselkedési
• (Structural) • Osztályok és objektumok összetétele	• (Creational) • Objektumok létrehozása	• (Behavioural) • Objektumok kölcsönhatása, interakciója

SINGLETON

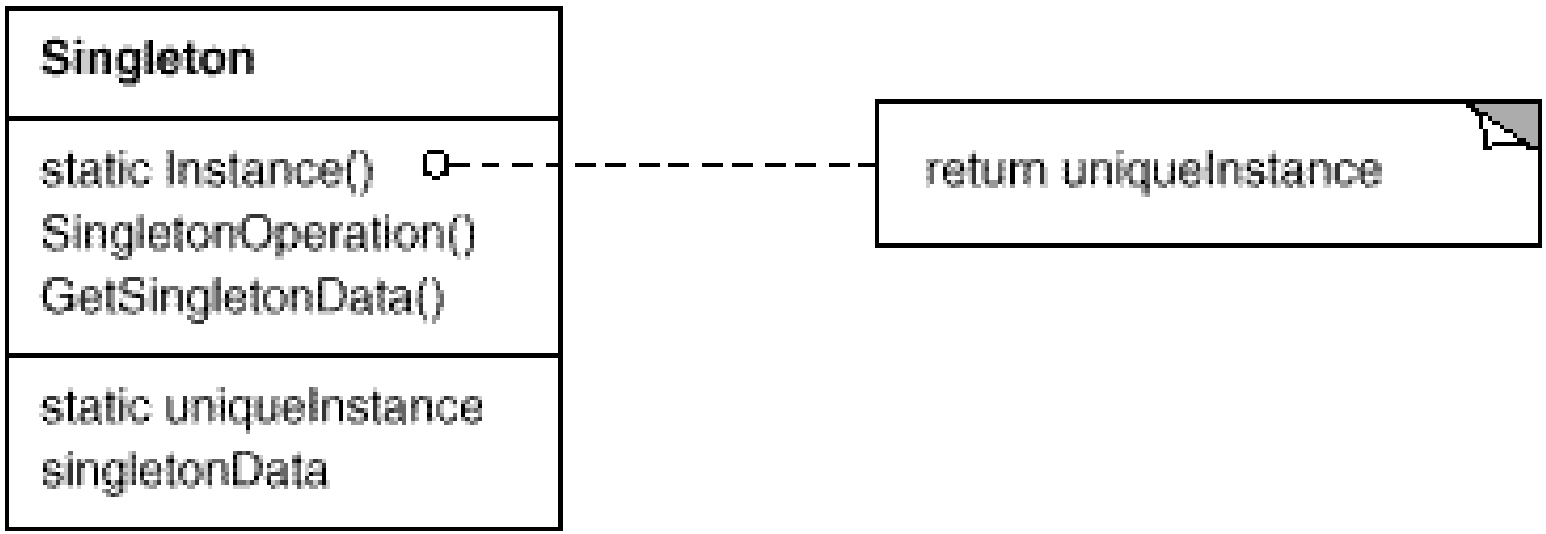
Egyke

Példa

- Beállítások
  - A beállítások elérése több helyről.
  - Mivel találjuk szemben magunkat?
    - Ha újra példányosítjuk az objektumot, akkor az már egy másik példány, nem lesz konzisztens.
    - Használhatunk globális változókat, de akkor például oda az öröklődés.

Singleton

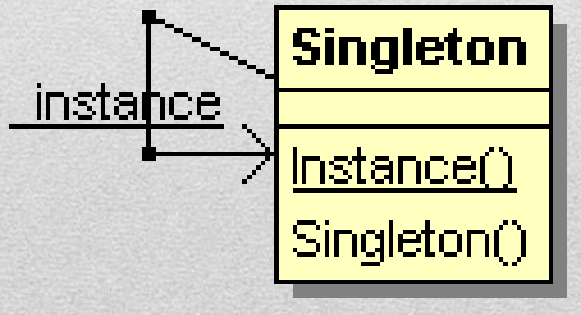
- ***Cél***: biztosítja, hogy egy osztályból csak egy objektum keletkezzen amely globálisan elérhető
- ***Alkalmazhatóság***:
  - pontosan egy példány létezhet amelyeket a kliensek elérhetnek
  - az egyedüli példány bővíthető kell hogy legyen (származtatással) amely ugyanúgy használható


Singleton
static Instance()
SingletonOperation()
GetSingletonData()
static uniqueInstance
singletonData
return uniqueInstance

Singleton

Singleton

- Hol használható?
  - Beállítások
  - Adatkapcsolatok (fájlrendszer, http, db)
  - Ablakkezelő
  - Naplózás
  - Factory



Különböző megvalósítások (Javaban)

- Globális változó
- Mohó és lusta kiértékelés
- Szálbiztos verzió
- Enum

- Például: java.lang.Runtime.getRuntime()

```java
package hu.u_szeged.inf.ovrt.singleton;

/**
 * Singleton with early instantiation. <br>
 * The singleton object is instantiated when the class <br>
 * is loaded and not when it is first used.
 */
public class Singleton {

    /** holds the single instance for the singleton class */
    private static final Singleton INSTANCE = new Singleton();

    /** private constructor -> no other instance can be created */
    private Singleton() {
        System.out.println("> create instance (!)");
    }

    /**
     * Access method to get the singleton instance.
     *
     * @return the singleton instance
     */
    public static Singleton getInstance() {
        return INSTANCE;
    }

    public void helloSingleton() {
        System.out.println("Hello! I'm a singleton.");
    }

    // test it
    public static void main(String[] args) {
        System.out.println("> program start");
        Singleton.getInstance().helloSingleton();
        System.out.println("> program finish");
    }
    // Output:
    // > create instance (!)
    // > program start
    // Hello! I'm a singleton.
    // > program finish
}
```

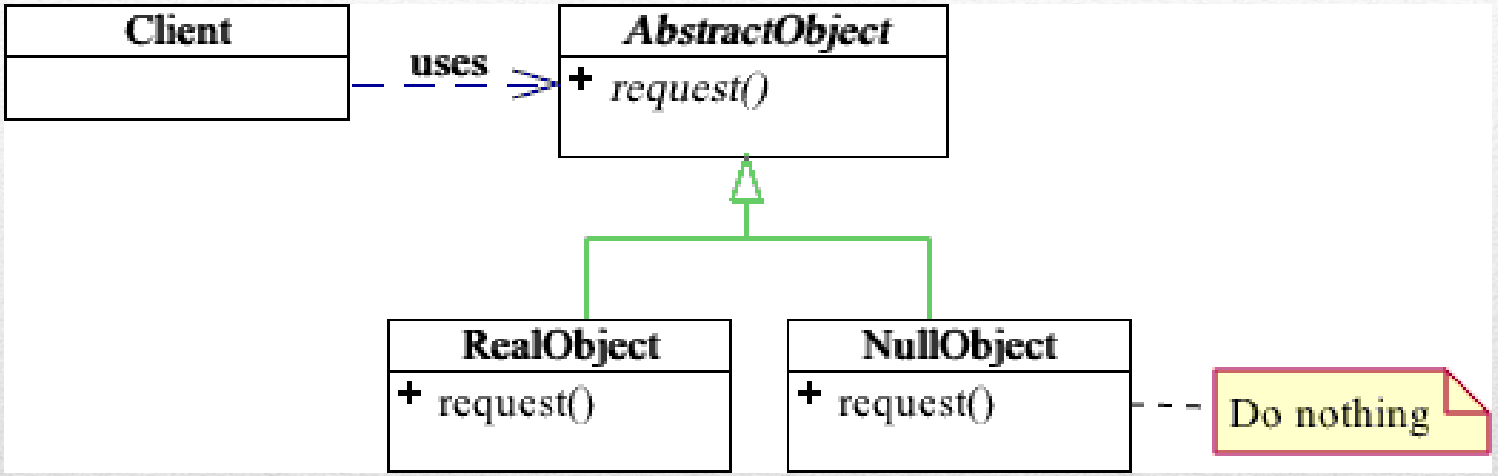
```java
package hu.u_szeged.inf.ovrt.singleton;

/**
 * Singleton with lazy instantiation.
 *
 * The singleton instance is created when the getInstance() <br>
 * method is called for the first time.
 */
public class SingletonLazy {

    private static SingletonLazy instance;

    private SingletonLazy() {
        System.out.println("> create instance (!)");
    }

    public static SingletonLazy getInstance() {
        if (instance == null) {
            // created only when needed
            instance = new SingletonLazy();
        }
        return instance;
    }

    public void helloSingleton() {
        System.out.println("Hello! I'm a singleton.");
    }

    public static void main(String[] args) {
        System.out.println("> program start");
        SingletonLazy.getInstance().helloSingleton();
        System.out.println("> program finish");
    }

    // Output
    // > program start
    // > create instance (!)
    // Hello! I'm a singleton.
    // > program finish

}
```

```java
public class SingletonLazyThreadSafe {

    private static SingletonLazyThreadSafe instance;

    private SingletonLazyThreadSafe() {
    }

    // ------------------- VERSION SIMPLE >> ----------------------
    // just use synchronized on the getter method
    // + simple method
    // - bad performance because of sync checking
    public static synchronized SingletonLazyThreadSafe getInstanceSimple() {
        if (instance == null) {
            instance = new SingletonLazyThreadSafe();
        }
        return instance;
    }
    // ------------------- << VERSION SIMPLE ----------------------

    // ------------------- VERSION BETTER >> ----------------------
    // only the creation of the instance is synchronized
    private static synchronized void createInstance() {
        if (instance == null) { // double check because of thread magic
            instance = new SingletonLazyThreadSafe();
        }
    }

    // no synchronized needed
    // - a bit more complex
    // + good performance, no sync checking needed
    public static SingletonLazyThreadSafe getInstanceBetter() {
        if (instance == null) {
            createInstance();
        }
        return instance;
    }
    // ------------------- << VERSION BETTER ----------------------

    public void helloSingleton() {
        System.out.println("Hello! I'm a singleton.");
    }
```

```java
package hu.u_szeged.inf.ovrt.singleton;

/**
 * Singleton implemented as an enum.
 *
 * + easy implementation <br>
 * + thread-safe <br>
 * ~ early initialization <br>
 * - no inheritance <br>
 */
public enum SingletonEnum {

    INSTANCE;

    public void helloSingleton() {
        System.out.println("Hello! I'm a singleton.");
    }

}
```

- https://stackoverflow.com/questions/137975/what-is-so-bad-about-singletons
- https://www.javaworld.com/article/2074979/java-concurrency/double-checked-locking--clever--but-broken.html

Singleton felvetések



Null object pattern