



Tema 3: Disseny

Anna Puig

Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona

Curs 2020/2021



UNIVERSITAT DE
BARCELONA

Temari

1	Introducció al procés de desenvolupament del software	
2	Anàlisi de requisits i especificació	
3	Disseny	
4	Del disseny a la implementació	
5	Ús de frameworks de testing	
		3.1 Introducció
		3.2 Patrons arquitectònics
		3.3 Criteris de Disseny: G.R.A.S.P.
		3.4 Principis de Disseny: S.O.L.I.D.
		3.5 Patrons de disseny

3.5. Patrons de disseny

Propòsit → Àmbit ↓	CREACIÓ	ESTRUCTURA	COMPORTAMENT
CLASSE	<ul style="list-style-type: none">• Factory method	<ul style="list-style-type: none">• class Adapter	<ul style="list-style-type: none">• Interpreter• Template method
OBJECTE	<ul style="list-style-type: none">• Abstract Factory• Builder• Prototype• Singleton• Object pool	<ul style="list-style-type: none">• Object Adapter• Bridge• Composite• Decorator• Facade• Flyweight• Proxy	<ul style="list-style-type: none">• Chain of Responsibility• Command• Iterator• Mediator• Memento• Observer• State• Strategy• Visitor

Patró Singleton (Idiom)



Nom: **Singleton**

Context

- Assegurar que una classe **només té una sola** instància i proporcionar un punt d'accés global a ella

Problema

- És necessari quan hi ha classes que han de gestionar de manera centralitzada un recurs
 - Un *spooler* d'impressió en un sistema
 - Un gestor de finestres, etc.
 - Controlador
- Una variable **global** no garanteix que només s'instancii una vegada

Patró Singleton: Exemple

```
public class Singleton {  
  
    private static Singleton uniqueInstance = new Singleton();  
  
    private Singleton() {}  
  
    public static Singleton getInstance() {  
        return uniqueInstance;  
    }  
  
    // other useful methods here  
    public String getDescription() {  
        return description+" de dins del mètode";  
    }  
}
```

Aquí hi ha **eager** initialization!

Patró Singleton: Exemple

```
public class SingletonClient {  
    public static void main(String[] args) {  
  
        System.out.println("Comença el main");  
  
        System.out.println(Singleton.description);  
  
        Singleton singleton = Singleton.getInstance();  
  
        System.out.println(singleton.getDescription());  
    }  
}
```

Quan i com es reserva la memòria de Singleton?

Patró Singleton: Exemple

```
public class Singleton {  
  
    public static String description = "I'm a statically initialized Singleton!";  
    private static Singleton uniqueInstance = new Singleton();  
    private Singleton() {}  
  
    public static Singleton getInstance() {  
        return uniqueInstance;  
    }  
  
    // other useful methods here  
    public String getDescription() {  
        return description+" de dins del mètode";  
    }  
}
```

Patró Singleton: Exemple

Aquí hi ha **eager** initialization!

```
public class SingletonClient {  
    public static void main(String[] args) {  
  
        System.out.println("Comença el main");  
  
        System.out.println(Singleton.description);  
  
        Singleton singleton = Singleton.getInstance();  
  
        System.out.println(singleton.getDescription());  
    }  
}
```

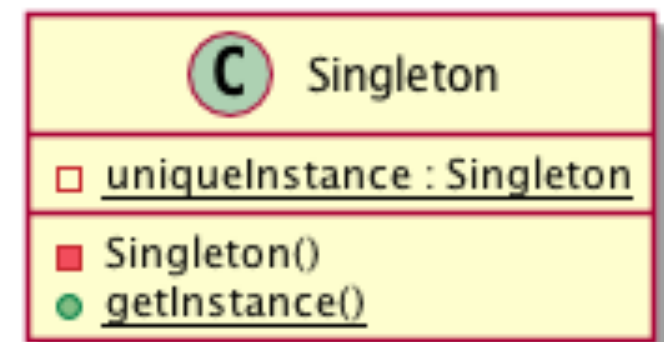
Output: **Comença el main**
I'm a statically initialized Singleton!
I'm a statically initialized Singleton! de dins del mètode

Patró Singleton

Solució

- És una classe on el constructor és **privat**
- Per a poder-se cridar i que es construeixi:
 - Es fa un mètode static (getInstance())
 - S'afegeix una variable estàtica i privada del mateix tipus de la classe on està continguda (instància)
- No es crea l'objecte fins que és necessari (**Lazy Initialization**)
- S'afegeix el codi necessari per no crear dues instàncies.

SINGLETON's Class Diagram



```
if (instancia== null )
    instancia= new Singleton()
return instancia;
```

Patró Singleton: Patró clàssic

```
public class Singleton {  
    private static Singleton uniqueInstance;  
  
    private Singleton() {}  
  
    public static Singleton getInstance() {  
        if (uniqueInstance == null) {  
            uniqueInstance = new Singleton();  
        }  
        return uniqueInstance;  
    }  
  
    // other useful methods here  
    public String getDescription() {  
        return "I'm a classic Singleton!";  
    }  
}
```

Aquí hi ha lazy initialization!

Patró Singleton

Nom del patró: **Singleton**

Consideracions:

- S'utilitza si es vol tenir una única instància d'una classe (per exemple d'un controlador, o una façana o una base de dades)
- S'utilitza Si es vol tenir accés global a unes dades des de diferents llocs de l'aplicació (per exemple: blackboards)

Pros:

- L'objecte Singleton només s'inicialitza la primera vegada que es crida.

Cons:

- Difícil de mantenir en entorns de multithreading
- Pot enmascarar alts acoblaments entre objectes

Patró Singleton: Múltiples threads no òptim

```
public class Singleton {  
    private static Singleton uniqueInstance;  
  
    // other useful instance variables here  
  
    private Singleton() {}  
  
    public static synchronized Singleton getInstance() {  
        if (uniqueInstance == null) {  
            uniqueInstance = new Singleton();  
        }  
        return uniqueInstance;  
    }  
  
    // other useful methods here  
    public String getDescription() {  
        return "I'm a thread safe Singleton!";  
    }  
}
```

Què passa amb múltiples threads?

Patró Singleton: Múltiples threads pseudo-òptim

```
public class Singleton {  
    private volatile static Singleton uniqueInstance;  
  
    private Singleton() {}  
  
    public static Singleton getInstance() {  
        if (uniqueInstance == null) {  
            synchronized (Singleton.class) {  
                if (uniqueInstance == null) {  
                    uniqueInstance = new Singleton();  
                }  
            }  
        }  
        return uniqueInstance;  
    }  
}
```

Volatile per optimitzar

Tots ells tenen problemes amb serialització i amb reflexió !

Patró Singleton: millor opció en Java

```
public enum Singleton {  
    INSTANCE;  
  
    // other useful attributes here  
  
    // other useful methods here  
    public String getDescription() {  
        return description+" de dins del mètode";  
    }  
}
```

```
public class SingletonClient {  
    public static void main(String[] args) {  
  
        Singleton singleton = Singleton.INSTANCE;  
  
        System.out.println(singleton.getDescription());  
    }  
}
```

Eager initialization, thread-safe, serializable

Patró Singleton: millor opció en Java

```
public enum Singleton {  
    INSTANCE("String concret");  
  
    // atributs privats  
  
    private Singleton(String text) {  
        // news dels atributs que calguin comprovant que són null abans de  
        // cridar al seus news  
    }  
  
    // other useful methods here  
}
```

```
public class SingletonClient {  
    public static void main(String[] args) {  
  
        Singleton singleton Singleton.INSTANCE;  
  
        System.out.println(singleton.getDescription());  
    }  
}
```

Eager initialization, thread-safe, serializable.
Només es pot cridar al constructor amb String
ja predefinit en el INSTANCE