Het Singleton Pattern

Analysis & design 2

katholieke hogeschool
associatie KU Leuven
vives

Singleton Pattern
Doel

- Singleton garandeert dat er van een klasse maar **1 instantie** kan bestaan

- Singleton zorgt voor een **globaal toegangspunt** voor die instantie

Singleton Pattern
Nut

Soms heeft een programma een object waar er maar 1 van nodig is:

- thread pool
- cache
- objecten die instellingen of registerwaarden bijhouden
- logger
- device driver
- factory
- …

Singleton Pattern
Opbouw

Neem een klasse:

```
public class SingletonClass
{
   public SingletonClass()
   { /* initialisatie */ }
}
```

Andere objecten mogen geen instanties kunnen maken. Hoe kunnen we de constructor afschermen?

Singleton Pattern
Opbouw

De constructor is afgeschermd als we hem private maken:

```
public class SingletonClass
{
   private SingletonClass()
   { /* initialisatie */ }
}
```

De enige plaats waar nu nog SingletonClass-instanties gemaakt kunnen worden, is in de klasse zelf.

Maar hoe kan die constructor aangeroepen worden zonder instantie?

Singleton Pattern
Opbouw

Een instantie is tijdelijk, maar een klasse bestaat altijd!

We moeten dus met een klassemethode (static) werken.

```
public class SingletonClass
{
    private SingletonClass()
    { /* initialisatie */ }

    public static SingletonClass getInstance()
    { /* unieke instantie retourneren */ }
}
```

Nu moet getInstance() nog uitgewerkt worden.

Singleton Pattern
Opbouw

```
public class SingletonClass
{
    private static SingletonClass uniqueInstance;

    private SingletonClass()
    { /* initialisatie */ }

    public static SingletonClass getInstance()
    {
        if (uniqueInstance == null)
            uniqueInstance = new SingletonClass();
        return uniqueInstance;
    }
}
```

Singleton Pattern
Multithreaded omgeving

In een **multithreaded** omgeving kunnen er met de voorgaande code toch meerdere instanties voorkomen!

Dat is omdat het dan mogelijk wordt dat twee threads tegelijk getInstance() uitvoeren.

Oplossingen?

Singleton Pattern
Threadsafe door synchronisatie

Oplossing 1: **synchronisatie**

Het kritische stuk van het pattern, de getInstance methode, wordt beschermd: er kan maar 1 thread tegelijk in.

```
public static synchronized SingletonClass
getInstance() { … }
```

Indien nodig moeten ook andere methodes beschermd worden!

Nadeel:

- resources nodig
- performantie van het programma kan verminderen

Singleton Pattern
Threadsafe door onmiddellijke instantiëring

Oplossing 2: **de unieke instantie aanmaken bij de programmastart**

De instantie wordt bij de programmastart gemaakt, en is daardoor gegarandeerd uniek.

```
private static SingletonClass uniqueInstance
    = new SingletonClass();
public static SingletonClass getInstance()
    { return uniqueInstance; }
```

Nadeel:

- Als de singleton-instantie veel resources eist, wil je de creatie ervan zo lang mogelijk uitstellen.

Singleton Pattern
Threadsafe door double checked locking

Oplossing 3: **synchronisatie, maar zo beperkt mogelijk**

```
private volatile static SingletonClass uniqueInstance;

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

Nadeel: wat ingewikkelder dan oplossing 1 en 2

Een andere aanpak
Enum

```
public enum SingletonClass {
   INSTANCE;

   // attributen en methodes
}
```

Vereist Java 1.5 of later.

Multiton
Meerdere singletons met gedeeld gedrag

```
public enum SingletonClass {
   INSTANCE1, INSTANCE2;

   // attributen en methodes
}
```

Singleton Pattern
Oefenen

Zet de klasse ChocolateBoiler om naar een threadsafe Singleton.

(de beginversie van de code staat op Toledo.)

Neem een klasse die je vroeger schreef, en maak er een threadsafe singleton van.
Wat moet je veranderen in de code die die klasse gebruikt?

Singleton pattern in .NET

Hierna volgen modelimplementaties van het Singleton Pattern in C# en Visual Basic. (Dit is geen examenstof.)

Let op de details bij het toepassen! (private, public, static, …)

Singleton Pattern in C#

Er wordt een readonly property gebruikt ipv een getter.
Deze oplossing is niet threadsafe:

```
public class Singleton
{
   private static Singleton instance;

   private Singleton() {}

   public static Singleton Instance {
      get {
         if (instance == null) {
            instance = new Singleton();
         }
         return instance;
      }
   }
}
```

Singleton Pattern in C#

Threadsafe met onmiddellijke instantiëring

```
public sealed class Singleton
{
   private static readonly Singleton instance = new Singleton();

   private Singleton(){}

   public static Singleton Instance
   {
      get
      {
         return instance;
      }
   }
}
```

Singleton Pattern in C#

Threadsafe met double checked locking

```
public sealed class Singleton
{
   private static volatile Singleton instance;
   private static object syncRoot = new Object();

   private Singleton() {}

   public static Singleton Instance {
      get {
         if (instance == null) {
            lock (syncRoot) {
               if (instance == null)
                  instance = new Singleton();
            }
         }

         return instance;
      }
   }
}
```

Singleton Pattern in Visual Basic.NET

Threadsafe!

```
Public Class Singleton

    Private Sub New()

    End Sub

    Public Shared ReadOnly Property GetInstance As Singleton
        Get
            Static Instance As Singleton = New Singleton
            Return Instance
        End Get
    End Property

End Class
```

Singleton Pattern
Samenvatting

- Een klasse met 1 unieke instantie

  private constructor

- Een globaal toegangspunt voor die instantie

  static getter