Design Patterns

MSc in Communications Software

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

Singleton

Design Pattern

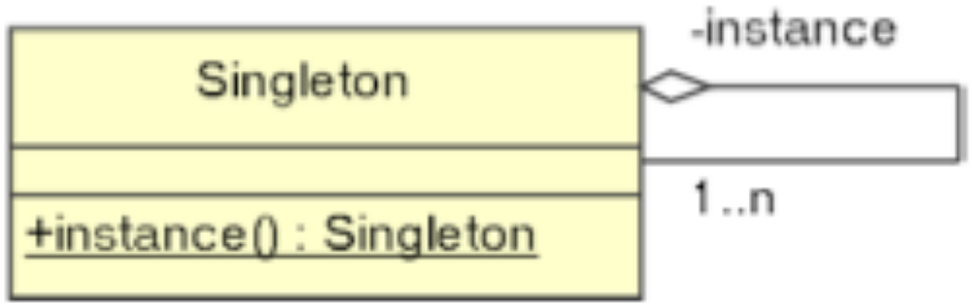
Intent

- A class with a constrained number of instances (typically one). The instance is globally accessible

- Singleton: The object being created, defines a class-level (static) get-instance method that returns the instance. The class-level get-instance method may create the object if necessary.

What Problem Does It Solve?

- Programs often have a need for single-instance objects.
- Objects, for example, might represent a single database, a single company, and so forth.



Advantages

- Better than a global object in that access is controlled and the global name space isn’t cluttered with hard-to-find objects.

- Guarantees that the object is created (and destroyed) only once—essential when the Singleton manages a global resource such as a database connection.

Disadvantages

- A Singleton called Globals that contains nothing but public variables can cause extreme difficulties

- Global namespace ‘polluted’ and difficult to trace impact of changes

    - A singleton containing global constants is reasonable if the values of the constants need to be initialized at run time.

    - If the values are known at compile time, use an interface made up solely of static final fields and implement the interface when you need to use the constants.

Implementation Variations (1)

```
class Singleton
{
  private static final Singleton instance = new Singleton();

  public static Singleton instance()
  {
    return instance;
  }
}
```

- Create single static instance in initializer
- Static ‘instance’ method to retrieve singleton

Example Implementation

```
class FileLogger
{
  private static FileLogger logger;

  // Prevent clients from using the constructor
  private FileLogger()
  {
  }

  // Control the accessible (allowed) instances
  public static FileLogger getFileLogger()
  {

    if (logger == null)
    {
      logger = new FileLogger();
    }
    return logger;
  }

  public synchronized void log(String msg)
  {
    // Write to log file...
  }
}
```

Implementation Variations (2)

- No non-static members
- All fields and members static

```
class Singleton
{
  static int all_fields;

  static void all_operations()
  {

  }
}
```

Implementation Variations (3) (holub)

- In a multi-threaded environment, more than one singleton may be created
- Mark instance method as ‘synchronized’ to prevent this.

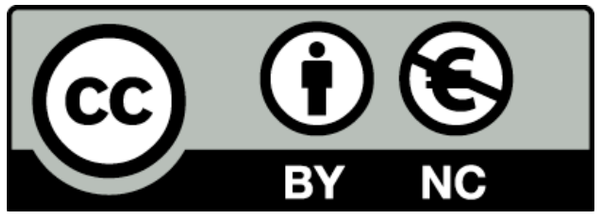
```
class Singleton
{
  private static Singleton1 instance;

  private Singleton1()
  {
    Runtime.getRuntime().addShutdownHook (
        new Thread()
        {
          public void run()
          {
           /* clean-up code here */
          }
      });
  }

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

JDK Examples

`Image picture = Toolbox.getDefaultToolbox().getImage(url);`	The **`Toolbox`** is a classic form of **`Singleton1`** in the Examples section. **`getDefaultToolbox()`** returns a **`Toolbox`** instance appropriate for the operating system detected at run time.
`Border instance = BorderFactory.createBevelBorder(3);`	Manages several **`Border`** instances, but only one instance of a **`Border`** object with particular characteristics (in this case, a 3-pixel beveled border) will exist, so it's a Singleton. All subsequent requests for a 3-pixel beveled border return the same object.
`Class class_object = Class.forName("com.holub.tools.MyClass");`	There's only one **`Class`** object for a given class, which effectively contain all static members.



Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit