Singleton Pattern: One of a Kind Objects

Singleton Pattern

- **Intent: Ensure that a class only has a single instance (which is accessed over a global point of access)**
  - MP3-File Player should only play one file at a time
    => single instance coordinates play-back
  - Database-Driver has to ensure global invariants
    => single instance coordinates DB access
  - Cache provides fast look-up for often used objects
    => single instance controls cache
  - Only one activity should access the camera
    => single instance coordinates access to the camera instance

Singleton Pattern

- **Example: Registry Class**

```
public class Registry {
   private Map<String, Object> entries =
      Collections.synchronizedMap(new HashMap<>());

   public Registry() { }

   public void register(String name, Object value) {
      ...
   }

   public Object lookup(String name) {
      ...
   }
}
```

  - Goal: One Instance of class `Registry` only, please!

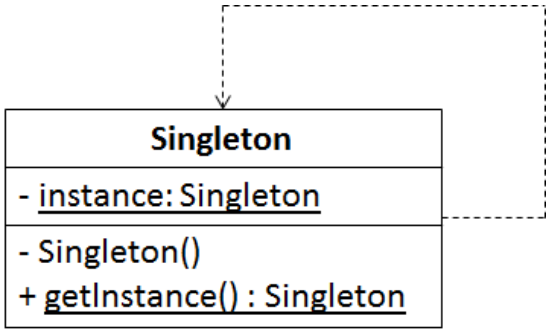
Singleton Pattern

- **Solution 1: Use static methods only**

```
public final class Registry {
   private static Map<String, Object> entries =
      Collections.synchronizedMap(new HashMap<>());

   private Registry() { }

   public static void register(String name, Object value) {
      ...
   }

   public static Object lookup(String name) {
      ...
   }
}
```

Singleton Pattern

- **Structure**



- **Code**

```
public final class Singleton {
   private Singleton() { }
   private static Singleton instance = new Singleton();
   public  static Singleton getInstance() {
      return instance;
   }
}
```

  - Private constructor prevents creation of instances outside of the class
  - Prevents creation of instances in subclasses as well => final

Singleton Pattern Example

```
public final class Registry {
   private Map<String, Object> entries =
      Collections.synchronizedMap(new HashMap<>());

   private Registry() { }
   private static Registry instance = new Registry();
   public  static Registry getInstance() {
      return instance;
   }

   public void register(String name, Object value) {
     ...
   }

   public Object lookup(String name) {
     ...
   }
}
```

```
Registry.getInstance().register("one", 1);
```

Singleton Pattern Samples

- java.lang.Runtime Runtime.getRuntime()
  - Every Java application has a single instance of class Runtime that allows the application to interface with the environment in which the application is running.
- java.lang.Class x.getClass()
  - Instances of the class *Class* represent classes and interfaces in a running Java application. Class *Class* has no public constructor; instances are constructed automatically by the JVM
  - Two instances of the same class refer to the same class instance
- java.util.logger.Logger Logger.getLogger(String name)
  - A Logger object is used to log messages for a specific system or application component
- java.awt.Taskbar Taskbar.getTaskbar()
  - The Taskbar class allows a Java application to interact with the system task area (taskbar, Dock, etc.).

Implementation Remarks

- **Enum**
  - Singleton may also be implemented as an enum

```
public enum SingletonDriver implements Driver {
   INSTANCE;
   public String toString() { return "Singleton"; }
   public void playSong(File file) { ... }
}
```

    - Advantages
      - Unique instance (access with `SingletonDriver.INSTANCE`)
      - Provides the serialization machinery for free
      - Interfaces may be implemented
    - Disadvantage
      - Fields are not serialized (only the name of the enum)
      - Cannot be extended to multiple instances

Singleton and Spring

- **Spring Singleton Beans**
  - By default all Spring beans are Singletons
- **Spring Prototype Beans**
  - Defining a prototype means instead of defining a single bean, one defines a blueprint
  - Bean instances are then created based on this blueprint

```
<bean id="person" class="ch.fhnw.Person" scope="prototype">
   …
</bean>
```

    - Every time the `getBean("person")` method is invoked a new instance of Person will be created

Relation with other Patterns

- **State**
  - State instances are often implemented as Singleton instances (could well be implemented using `enums`)
- **Abstract Factory**
  - This pattern can use a Singleton for providing the current factory
- **Façade**
  - The façade objects are often Singletons because only one instance is required

Singleton Pattern: 15 Years Later

- **When discussing which patterns to drop, we found that we still love them all. (Not really—I'm in favor of dropping Singleton. Its use is almost always a design smell.)**
  - Erich Gamma
    Design Patterns 15 Years Later
    http://www.informit.com/articles/article.aspx?p=1404056

- **Singletons are often used as a justification for global state. Easy to add / difficult to remove**
  - Erich Gamma
    Design Patterns: Past, Present and Future
    FOSE (The Future of Software Engineering Symposium) 2010
    http://fose.ethz.ch/slides/gamma.pdf

Singleton Disadvantages

- **Hidden coupling from potentially everywhere!**
  - Singleton provides a global access point to a service, but this coupling not visible by examining the interfaces of the classes that use the Singleton
- **Violation of the Single Responsibility Principle**
  - A Singleton allows to limit the creation of objects, which means that two responsibilities are mixed together into one class:
    - Its own singularity
    - Its functionality
  - http://c2.com/cgi/wiki?SingleResponsibilityPrinciple

Singleton Disadvantages

- **A Singleton promotes tight coupling between classes**
  - Problem: testing. A Singleton object prevents the polymorphic substitution of another, simpler object (mock object)
  - A better solution is (once more) to delegate the creation of the object to, e.g., a simple Factory.
  - Or: Base your code onto the principle of Dependency Injection and use Spring, some other DI-framework or provide your own mechanism.

- **Singletons carry state**
  - Problem: testing,
    - Singleton object is created before the first test uses it
    - The *same* Singleton is reused all over the time in any other test, being perhaps in some weird state!

Discussion and Comments

- **What is so bad about Singletons?**

What is so bad about singletons? [closed]

1387

The singleton pattern is a fully paid up member of the GoF's patterns book, but it lately seems rather orphaned by the developer world. I still use quite a lot of singletons, especially for factory classes, and while you have to be a bit careful about multithreading issues (like any class actually), I fail to see why they are so awful.

Stack Overflow especially seems to assume that everyone agrees that Singletons are evil. Why?

695

design-patterns singleton

  - 36 Answers can be found at http://stackoverflow.com/questions/137975/what-is-so-bad-about-singletons