Design Patterns - Creational- Singleton

Saturday, May 21, 2016 2:24 PM

Describes the patterns in which objects are created.

Types of Creational Patterns:

- Singleton
- Builder
- Prototype
- Factory
- Abstract Factory

What we will cover:

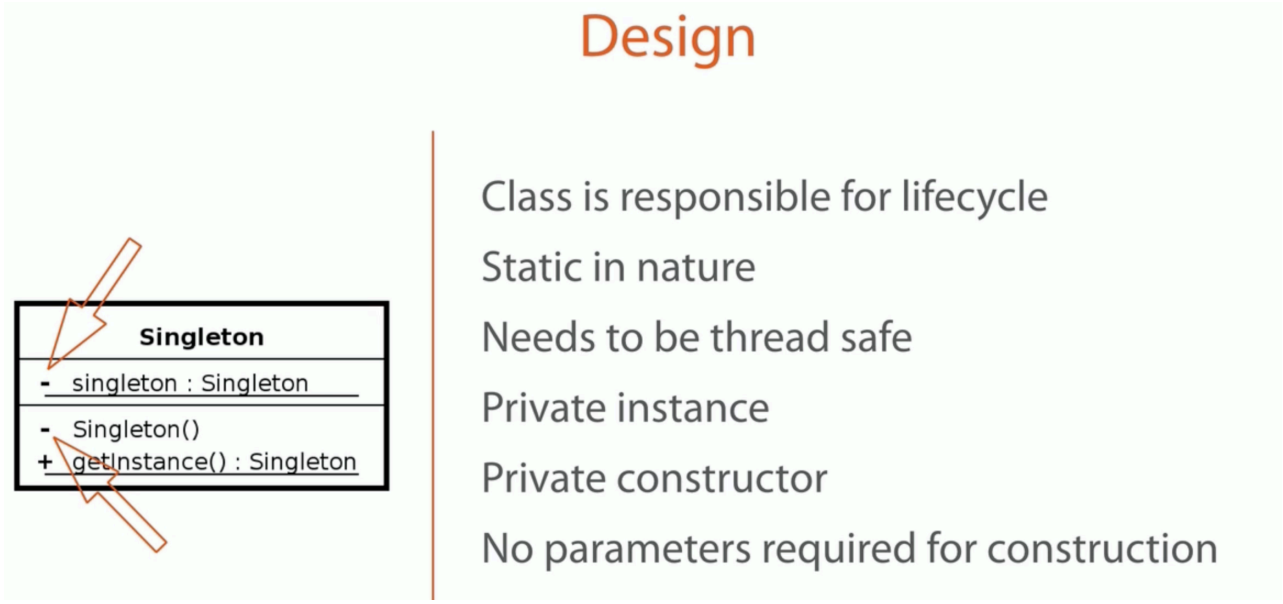
Overview
Concepts
Design
Live Example in Java
Demo
Pitfalls
Contrast

Singleton Pattern:

- Only one instance
- Lazily Loaded
    - Examples
        - Logger
        - Spring Beans

Key Points:

- Will have PRIVATE - NO ARGUMENT Constructor , if it has then it violates Singleton and will become FACTORY PATTERN



Demo:

```
package info.javagrasp.singleton;

public class SingletonEveryDayBean {

    public static void main(String[] args){
    Runtime singletonRunTime = Runtime.getRuntime();
    System.out.println(singletonRunTime.hashCode());
    Runtime secondInstanceOfSingleton = Runtime.getRuntime();
    System.out.println(secondInstanceOfSingleton.hashCode());
    }
}
```

Problems @ Javadoc Declaration Console

```
<terminated> SingletonEveryDayBean [Java Application] /Library/Java/JavaVirtualMachines/jd
2018699554
2018699554
```

Steps to create Singleton:

1) Create a new class.
2) Create a PRIVATE STATIC instance of the same class inside it.
3) Create a PRIVATE no args constructor. So that only we can create an object of that type.
4) Create a PUBLIC STATIC getter that will provide this PRIVATE instance

Eagerly Loaded:

Means when the JVM starts it will create this object

Dashboard | DBSingleton.java | SingletonEveryDayBean.java

```
package info.javagrasp.singleton;

public class DBSingleton {

    private static DBSingleton instance = new DBSingleton();

    private DBSingleton(){

    }

    public static DBSingleton getInstance(){
        return instance;
    }
}
```

DBSingletonDemo.java

```
package info.javagrasp.singleton;

public class DBSingletonDemo {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        DBSingleton instance = DBSingleton.getInstance();
        System.out.println(instance.hashCode());
        DBSingleton _2instance = DBSingleton.getInstance();
        System.out.println(_2instance.hashCode());
    }

}
```

Lazy Loaded:

Only when asked for it will be created, so that the startup time will be fast

Dashboard | DBSingleton.java | SingletonEveryDayBean.java

```
package info.javagrasp.singleton;

public class DBSingleton {

    //Eagerly Loaded
    //private static DBSingleton instance = new DBSingleton();

    //Lazy Loaded
    private static DBSingleton instance = null;

    private DBSingleton(){

    }

    public static DBSingleton getInstance(){
        //Lazy Loading
        if(null == instance){
            instance = new DBSingleton();
        }
        return instance;
    }
}
```

DBSingletonDemo.java

```
package info.javagrasp.singleton;

public class DBSingletonDemo {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        DBSingleton instance = DBSingleton.getInstance();
        System.out.println(instance.hashCode());
        DBSingleton _2instance = DBSingleton.getInstance();
        System.out.println(_2instance.hashCode());
    }

}
```

Thread Safe:

Making it synchronized will actually slowdown but will be thread safe (while writing this notes I don't have any idea what this means) but it's a good practice

Dashboard | DBSingleton.java | SingletonEveryDayBean.java

```
package info.javagrasp.singleton;

public class DBSingleton {

    //Eagerly Loaded
    //private static DBSingleton instance = new DBSingleton();

    //Lazy Loaded
    private static DBSingleton instance = null;

    private DBSingleton(){
```

DBSingletonDemo.java

```
package info.javagrasp.singleton;

public class DBSingletonDemo {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        DBSingleton instance = DBSingleton.getInstance();
        System.out.println(instance.hashCode());
        DBSingleton _2instance = DBSingleton.getInstance();
        System.out.println(_2instance.hashCode());
    }
```

```
    }

    public static DBSingleton getInstance(){
        //Lazy Loading
        if(null == instance){
            //thread safe
            synchronized (DBSingleton.class) {
                if(null == instance){
                    instance = new DBSingleton();
                }
            }
        }
        return instance;
    }
}
```

```
    }
```