Introduction to Design Patterns & Singleton Pattern

Lecture-2

Design Patterns

A pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem

Software Structural Problems

- Abstraction,
- Encapsulation
- Information hiding
- Separation of concerns
- Coupling and cohesion
- Separation of interface and implementation
- Single point of reference
- Divide and conquer

Non Functional Problems

- Changeability
- Interoperability
- Efficiency
- Reliability
- Testability
- Reusability

The “gang of four” - GOF

Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides

(Addison-Wesley, 1995)

Types of Patterns

Creational

address problems of creating an object in a flexible way. Separate creation, from operation/use

Structural

address problems of using O-O constructs like inheritance to organize classes and objects

Behavioral

address problems of assigning responsibilities to classes. Suggest both static relationships and patterns of communication

Types of Patterns

Creational

- Singleton
- Abstract factory
- Factory
- Prototype
- Builder

Structural

- Adapter
- Decorator
- Bridge
- Façade
- Composite
- Proxy

Behavioral

- Command
- State
- Template
- Observer
- Strategy
- Chain of Responsibilty
- Mediator
- Visitor

Singleton Pattern

- Singleton pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the Java Virtual Machine.
- The singleton class must provide a global access point to get the instance of the class.

Singleton Pattern Implementation

- Private constructor to restrict instantiation of the class from other classes.
- A private static variable of the same class that holds the only instance of the class.
- Public static method that returns the instance of the class, this is the global access point for the outer world to get the instance of the singleton class.

Eager Initialization

Copy

```
package com.journaldev.singleton;

public class EagerInitializedSingleton {

    private static final EagerInitializedSingleton instance = new EagerInitializedSingleton();

    // private constructor to avoid client applications using the constructor
    private EagerInitializedSingleton(){}

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

Static Block Initialization

Copy

```
package com.journaldev.singleton;

public class StaticBlockSingleton {

    private static StaticBlockSingleton instance;

    private StaticBlockSingleton(){}

    // static block initialization for exception handling
    static {
        try {
            instance = new StaticBlockSingleton();
        } catch (Exception e) {
            throw new RuntimeException("Exception occurred in creating singleton instance");
        }
    }

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

Lazy Initialization

```
package com.journaldev.singleton;

public class LazyInitializedSingleton {

    private static LazyInitializedSingleton instance;

    private LazyInitializedSingleton(){}

    public static LazyInitializedSingleton getInstance() {
        if (instance == null) {
            instance = new LazyInitializedSingleton();
        }
        return instance;
    }
}
```

Thread Safe Initialization

```
package com.journaldev.singleton;

public class ThreadSafeSingleton {

    private static ThreadSafeSingleton instance;

    private ThreadSafeSingleton(){}

    public static synchronized ThreadSafeSingleton getInstance() {
        if (instance == null) {
            instance = new ThreadSafeSingleton();
        }
        return instance;
    }

}
```

Thread Safe Initialization – Better Approach

```
public static ThreadSafeSingleton getInstanceUsingDoubleLocking() {
    if (instance == null) {
        synchronized (ThreadSafeSingleton.class) {
            if (instance == null) {
                instance = new ThreadSafeSingleton();
            }
        }
    }
    return instance;
}
```

Singleton Pattern Usage

- Logging
- Hardware access
- Database connections
- Config files
- Driver objects