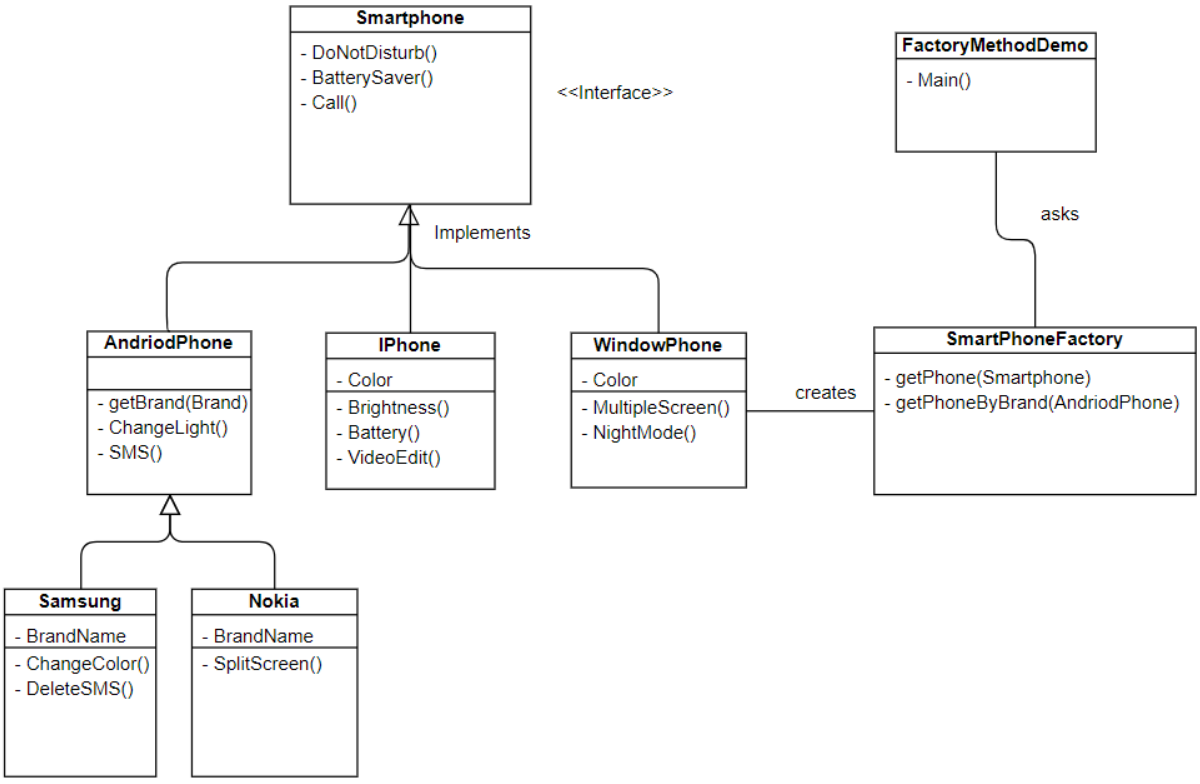
Ayushi Garg

ayushi.garg@accolitedigital.com

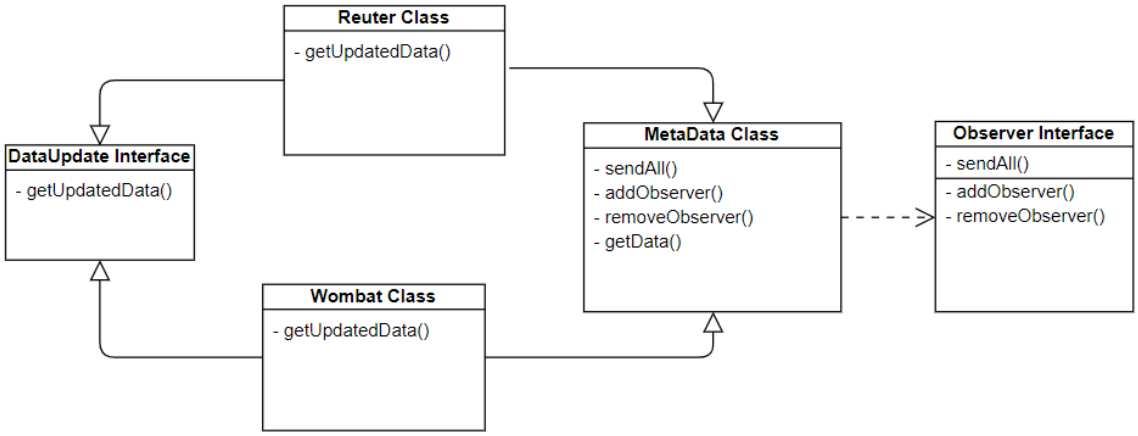
1. You have a Smartphone class and will have derived classes like IPhone, AndroidPhone, WindowsMobilePhone can be even phone names with brand, how would you design this system of Classes.

Solution – The above situation can be solved using Factory pattern as the object of each phone is created without showing the implementation to the end user.



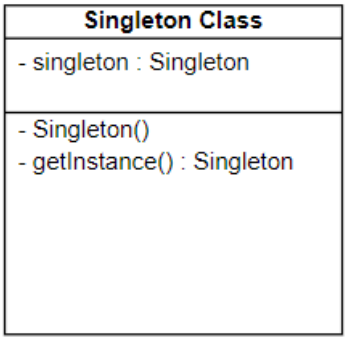
2. Write classes to provide Market Data and you know that you can switch to different vendors overtime like Reuters, wombat and may be even to direct exchange feed , how do you design your Market Data system.

Solution – The above situation is solved using strategy method as it requires the decision to shift to different vendor for the data.



3. What is Singleton design pattern in Java ? write code for thread-safe singleton in Java and handle Multiple Singleton cases shown in slide as well.

Solution - The singleton pattern is a software design pattern that restricts the instantiation of a class to one "single" instance. This is useful when exactly one object is needed to coordinate actions across the system.



This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

Thread-safe version of Singleton

```
package com.java.design.pattern.singleton;

// Thread-Safe Implementation
public class SingletonClassDemo3 {
   private static SingletonClassDemo3 instance = null;
   private SingletonClassDemo3() {
   }
   public static synchronized SingletonClassDemo3 getInstance() {
```

```
    if (instance == null) {
      instance = new SingletonClassDemo3();
    }
    return instance;
}}
```

Ways to handle Multiple Singleton Cases –

1. **Reflection**

   Reflection can easily destroy the Singleton design of a class by calling the private constructor and setting the access level to true.

```
try {
Constructor[] constructors = Singleton.class.getDeclaredConstructors();
        for (Constructor constructor : constructors) {

    // Below Code Will Destroy the Singleton Pattern
    constructor.setAccessible(true);
     instanceTwo = (Singleton) constructor.newInstance();
    break;
}
    } catch (Exception ex) {
        ex.printStackTrace();
    }
```

   To overcome this, Enum is used because JVM ensures that the Enum value is instantiated only once and the objects returned by Enum are Singleton in nature. The purpose of using Enum is that its default constructor is private in nature and developers cannot invoke them through the program.

2. **Serialization**

   In distributed systems, the Singleton design can be destroyed during the deserialization process as it'll create a new instance of the Singleton class. To overcome this issue, developers have to implement the `readResolve()` method in the Singleton class implementing the `Serializable` interface.

```
    // Implementing the 'readResolve()' method.
        protected Object readResolve() {
            return getInstance();
        }
```

4. Design classes for Builder Pattern.

Solution –