Design Patterns Assignment

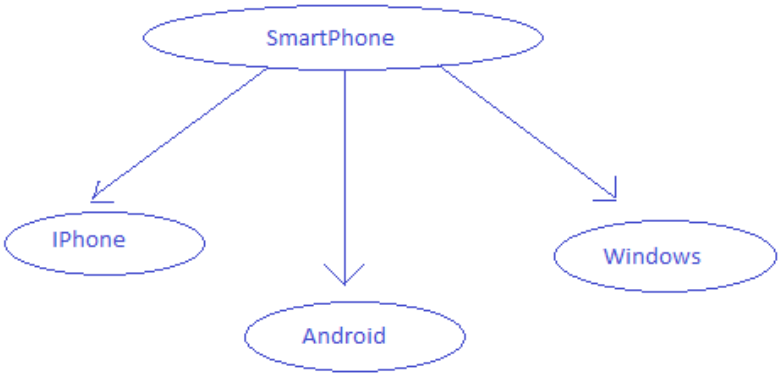
1. You have a Smartphone class and will have derived classes like IPhone, AndroidPhone,WindowsMobilePhone can be even phone names with brand, how would you design this system of Classes

   This system of classes can be best designed using Factory Method. It solves the problem of creating objects without specifying the concrete classes.

   Abstract Parent Class - SmartPhone

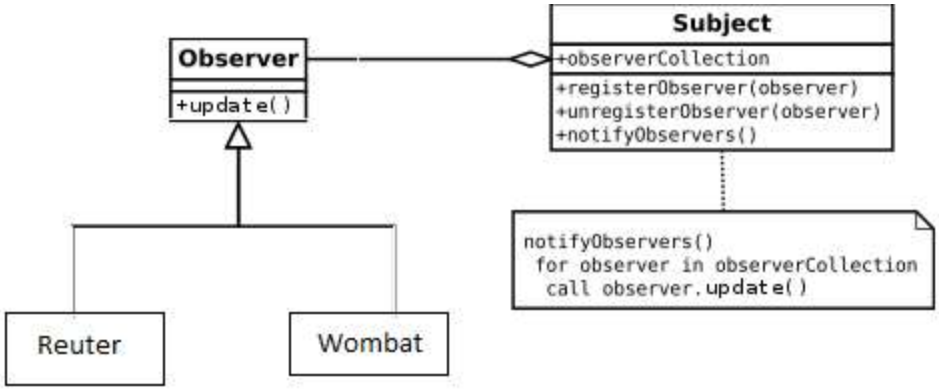
   All Other Phones will be subclasses extending the SmartPhone class.

```
public abstract class Smartphone{
 public void getConfig() { }

 public void getBattery() { }
}

public Class SmartPhoneFactory{
 public static Phone getPhoneObject(String phone) {
  Smartphone ph;
  if("Iphone".equals(option)) ph=new Iphone();
  if("Android".equals(option)) ph=new Android();
  else ph=new WindowsPhone();
}
}

public class SmartPhoneTest{
 public static void(String args[]) {
  String option=args[0];
   SmartPhone phone=SmartPhoneFactory.getPhoneObject(option);
  }
 phone.getConfig();
 phone.getBattery();
}
```



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.

Observer Design Pattern is the best pattern as Observer design pattern is based on communicating changes in state of object to observers so that they can take the appropriate action.



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

The singleton involves only one class which is responsible to instantiate itself, to make sure it creates not more than one instance. In the same time it provides a global point of access to that instance. In this case, the same instance can be used from everywhere making it

impossible to invoke the constructor directly each time.

Thread Safe Singleton

```
public class ThreadSafe
{
  private static ThreadSafe obj;

  private ThreadSafe()
  {
  }

  synchronized public static ThreadSafe getObj()
  {
    if (obj== null)
    {
      obj= new ThreadSafe();
    }
    return obj;
  }
}
```

Eager Initialization

```
public class Eager
{

  private static final Eager obj = new Eager();

  private obj()
  {
    // private constructor
  }
  public static Eager getObj(){
        return obj;
    }
}
```

Lazy initialization

```
public class Lazy
{
  private static Lazy obj;
  private Lazy()
  {
  }

  public static Lazy getObj()
```

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

4. Design classes for Builder Pattern