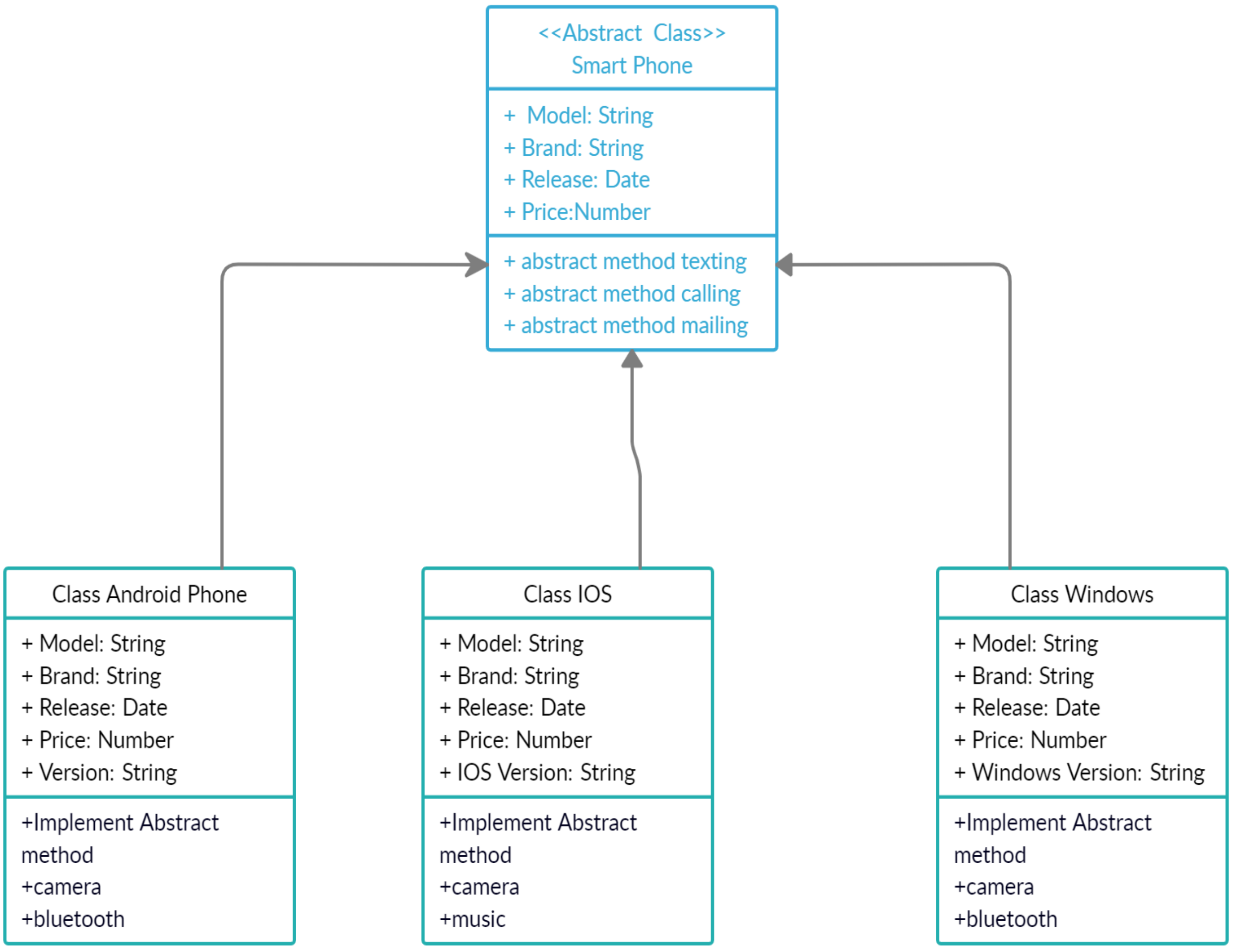
Spring AU -Design Principles and Patterns-Anubhav Singh

Make a Class diagram for below 4 scenarios and Name the Design pattern used.

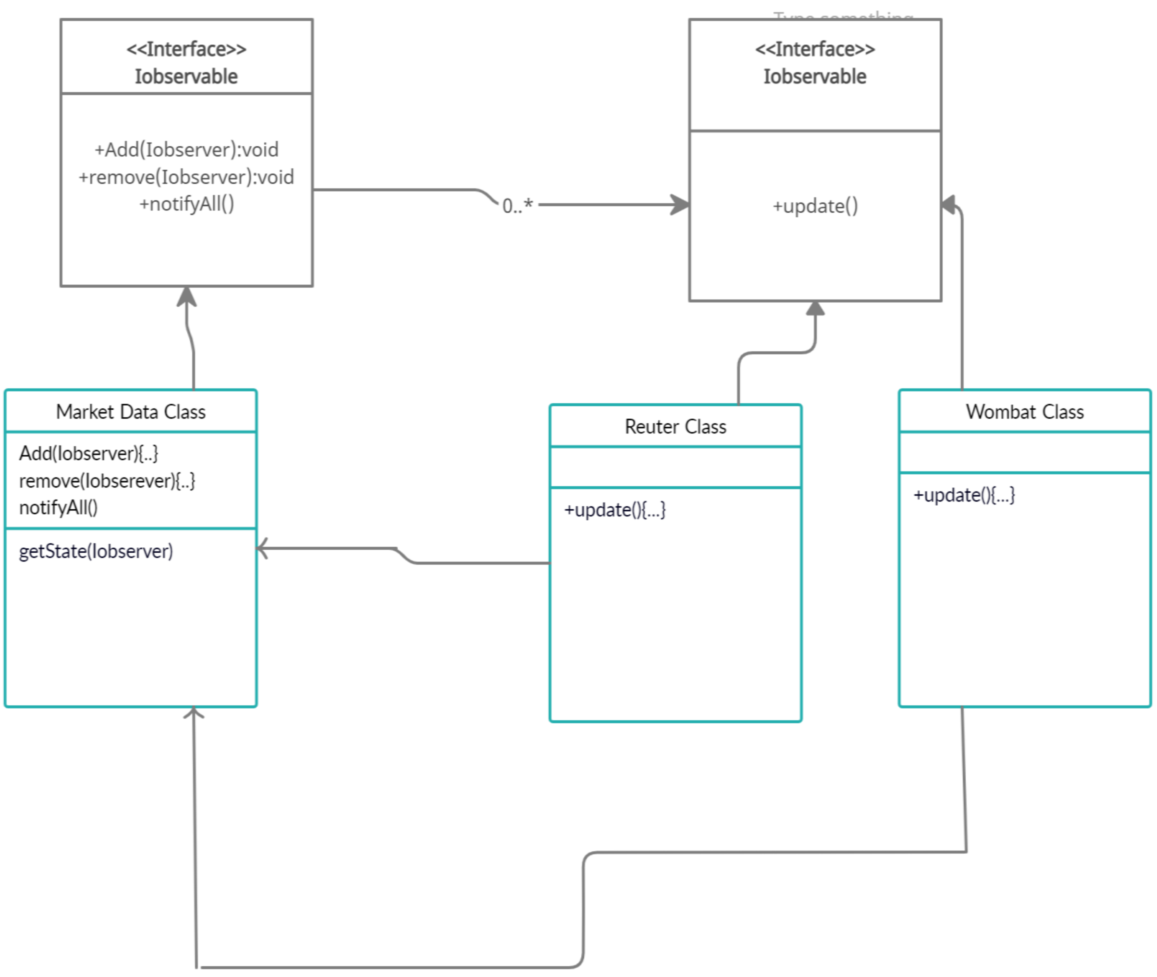
Q1) You have a Smartphone class and will have derived classes like IPhone, Android Phone, Windows Mobile Phone
can be even phone names with brand, how would you design this system of Classes.

The "Strategy Pattern" seems most suitable for the above scenario . The Smartphone class can be made an abstract class and other class can just be extended from the abstract class.



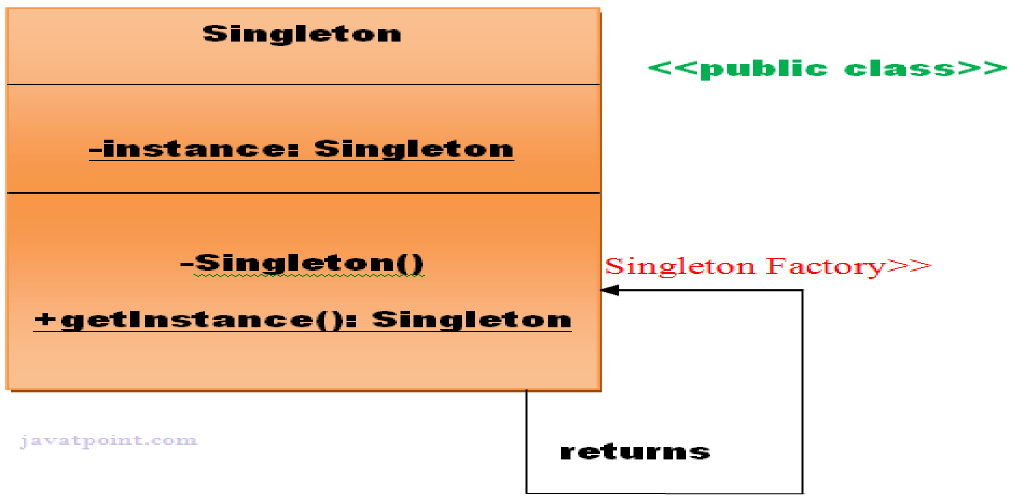
Q2) Write classes to provide Market Data and you know that you can switch to different vendors overtime like Reuters, wombat and may be even to direct exchange feed, how do you design your Market Data system.

In this case the “Observer Design Pattern” seems most suitable.The updated price stocks can be pushed to the vendors.



Q3) 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 design pattern is used to restrict the instantiation of a class and ensures that only one instance of the class exists in the JVM. In other words, a singleton class is a class that can have only one object (an instance of the class) at a time per JVM instance.



There are two forms of singleton design pattern

- **Early Instantiation:** creation of instance at load time.
- **Lazy Instantiation:** creation of instance when required.

Advantage of Singleton design pattern

- Saves memory because object is not created at each request. Only single instance is reused again and again.

Usage of Singleton design pattern

- Singleton pattern is mostly used in multi-threaded and database applications. It is used in logging, caching, thread pools, configuration settings etc.

In general, we follow the below steps to create a singleton class:

1. Create the private constructor to avoid any new object creation with new operator.
2. Declare a private static instance of the same class.
3. Provide a public static method that will return the singleton class instance variable. If the variable is not initialized then initialize it or else simply return the instance variable.

MULTITHREADING SOLUTION

```
public class Singleton {
     private static Singleton uniqueInstance;

        // other useful instance variables here

      private Singleton() {}
    public static synchronized Singleton
getInstance() {
   if (uniqueInstance==null){
        uniqueInstance = new Singleton();}
        return uniqueInstance;
}
//other useful methods here
}
```

Singleton Class in Java using Early Loading

```
/**
* Singleton Class in Java with Early loading
*/
public class SingletonCls {

    private static final SingletonCls singletonInst =
new SingletonCls();
    private String message = "";
    private SingletonCls() {
    }
    public static SingletonCls getInstance(){
        return singletonInst;
    }
    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

}
```

Singleton Class in Java using Lazy Loading

```
/**
* Singleton Class in Java with Lazy loading
*/
public class SingletonCls {
    private static SingletonCls singletonInst;
    private String message = "”;
    private SingletonCls() {
        System.out.println("Singleton instance
created.");
    }
    public static SingletonCls getInstance(){
        if(singletonInst==null){
            singletonInst = new SingletonCls();
            System.out.println("SingletonCls instance
created for the first time.");
        }
        return singletonInst;
    }

    // getter for the variable message
    public String getMessage() {
        return message;
    }

    // setter for the variable message
    public void setMessage(String message) {
        this.message = message;
    }
}
```

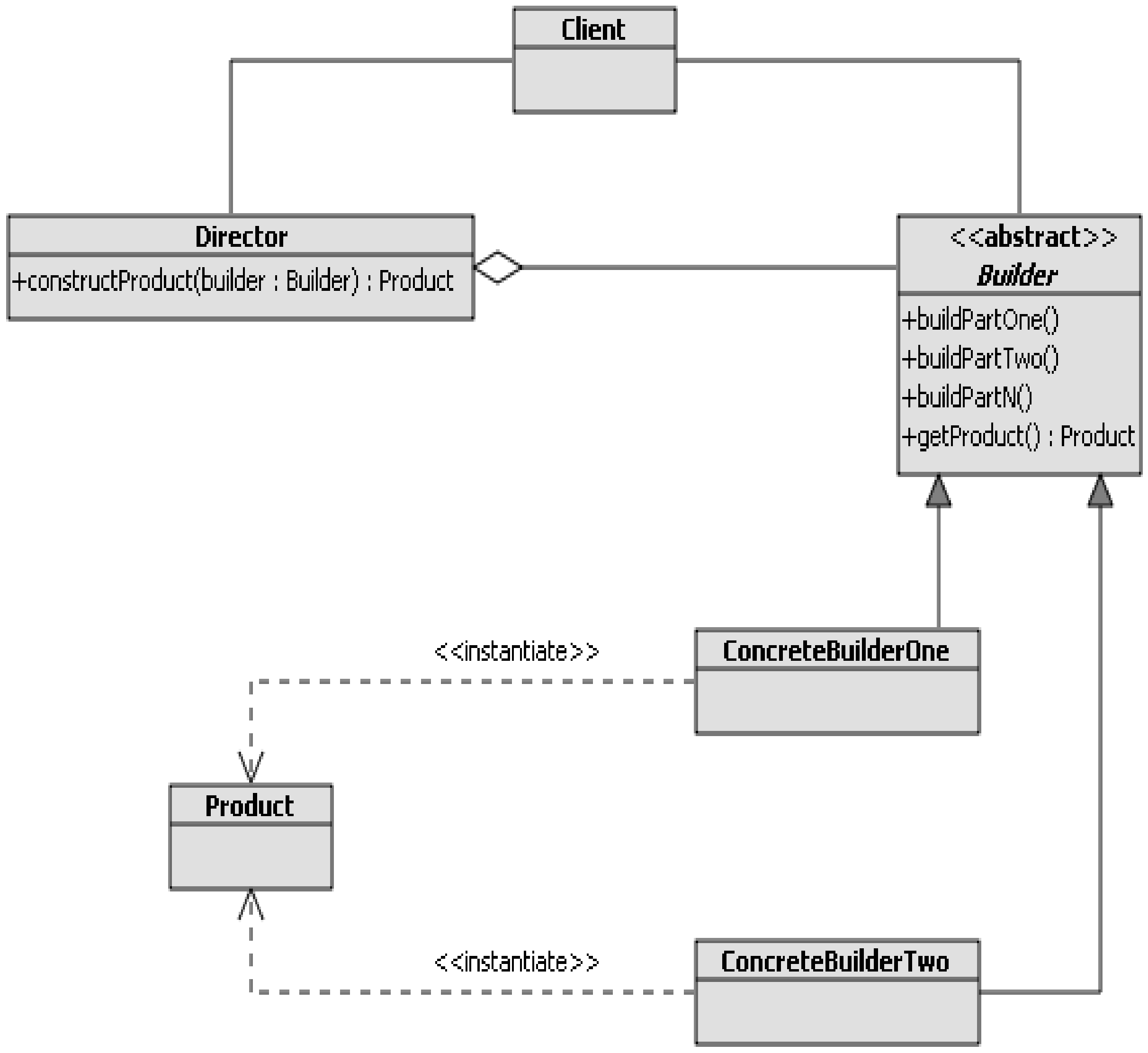
Singleton Class in Java using Lazy Loading that is thread safe

The instance could be initialized only when the Singleton Class is used for the first time. To make the creation of Singleton instance thread safe, the method getInstance() is synchronized so that the method is executed by only one thread at a time.

```
/**
* Singleton Class in Java with Lazy loading
*/
public class SingletonCls {
    // singleton instance declaration
    private static SingletonCls singletonInst;

    // a variable of singleton class
    private String message = "";

    // making constructor private so that no other
class could use the default constructor
    private SingletonCls() {
        System.out.println("Singleton instance
created.");
    }

    // the method which gives access to the only
instance of SingletonCls, is thread safe
    public static synchronized SingletonCls
getInstance(){
        if(singletonInst==null){
            singletonInst = new SingletonCls();
            System.out.println("SingletonCls instance
created for the first time.");
        }
        return singletonInst;
    }

    // getter for the variable message
    public String getMessage() {
        return message;
    }

    // setter for the variable message
    public void setMessage(String message) {
        this.message = message;
    }
}
```

Q4) Design classes for Builder Pattern.

Builder pattern aims to “Separate the construction of a complex object from its representation so that the same construction process can create different representations.” It is used to construct a complex object step by step and the final step will return the object. The process of constructing an object should be generic so that it can be used to create different representations of the same object.


Client
Director
+constructProduct(builder : Builder) : Product
<<abstract>>
Builder
+buildPartOne()
+buildPartTwo()
+buildPartN()
+getProduct() : Product
<<instantiate>>
ConcreteBuilderOne
Product
<<instantiate>>
ConcreteBuilderTwo