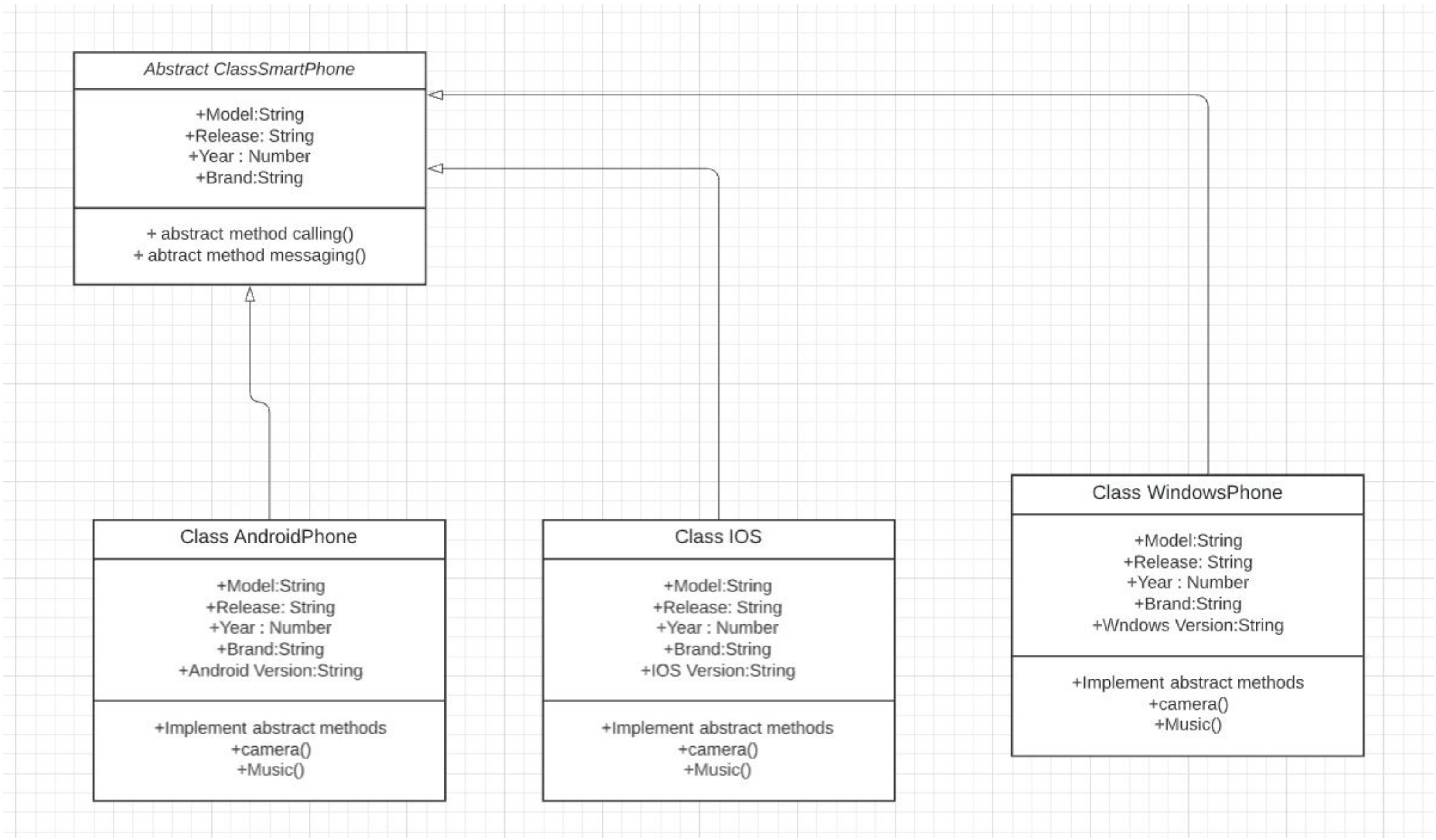
sumanto.pal@accolitedigital.com
Sumanto Pal

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

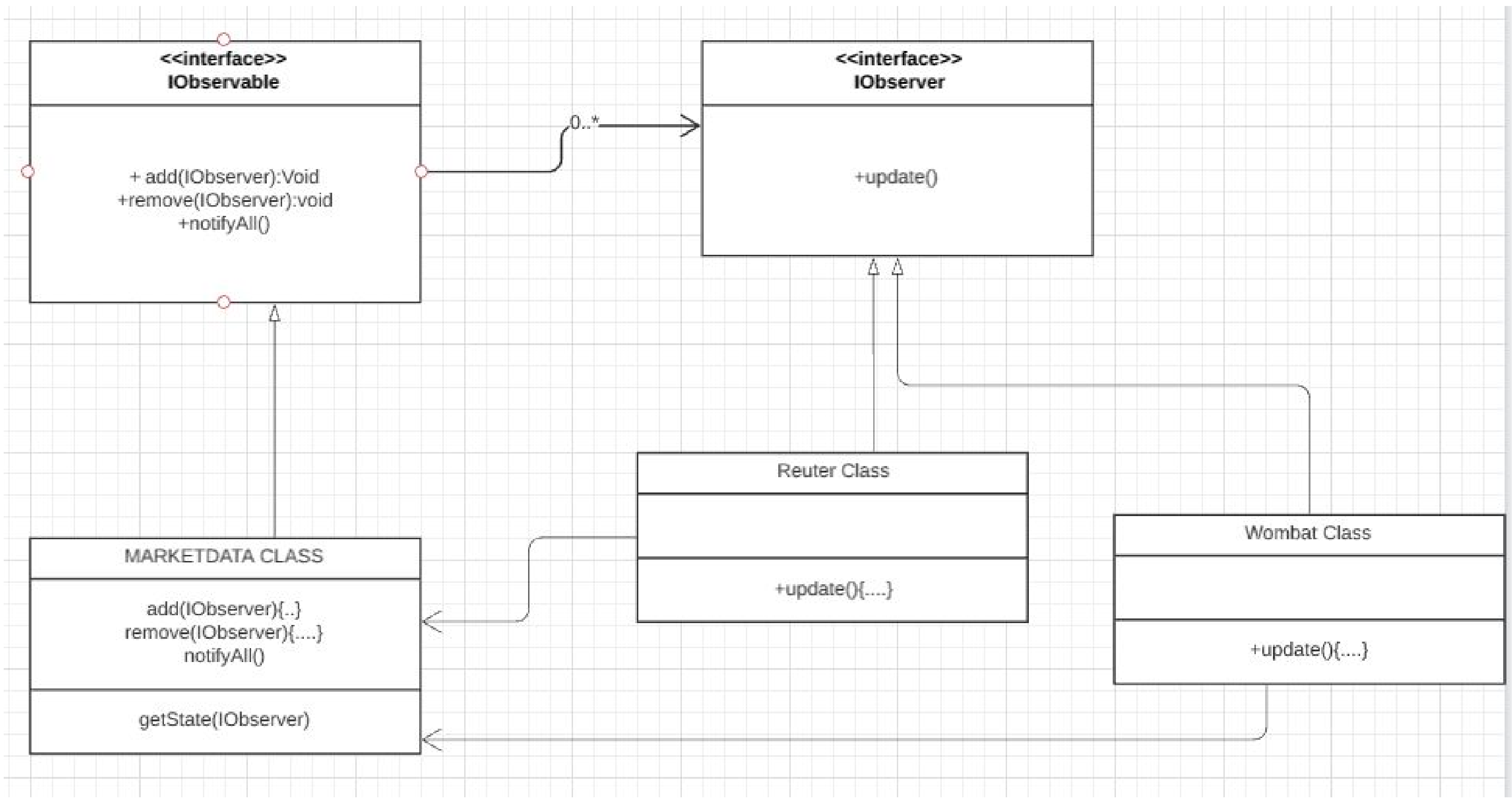
1. You have a Smartphone class and will have derived classes like IPhone, AndroidPhone,WindowsMobilePhone can be even phone names with brand, how would you design this system of Classes.

Ans: For the above scenario, The strategy Pattern seems most suitable, by Making SmartPhone Class as Abstract and other Class extending it.



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

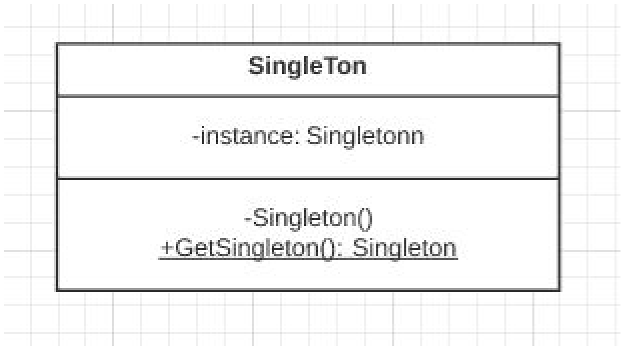
Ans: For the above scenario, The Observer Design Pattern seems most suitable, by pushing all the update price stocks to its Vendors



PSEUDOCODE LINK→ https://ideone.com/qNmZC1

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.

ANS:



DEFINITION :

→ The Singleton Pattern ensure a class has only one instance, and provides a global point of access to it

WHEN TO USE:

→ There are many objects we only need one of: thread pools, caches, dialog boxes, objects that handle preferences and registry settings, objects used for logging, and objects that act as device drivers to devices like printers and graphics cards. In fact, for many of these types of objects, if we were to instantiate more than one we'd run into all sorts of problems like incorrect program behavior, overuse of resources, or inconsistent results.

→ The Singleton Pattern is a time-tested method for ensuring only one object gets created. The Singleton Pattern also gives us a global point of access, just like a global variable, but without the downsides.

ISSUES:

--> In an multi-threaded environment, Race condition may lead to creation of more than one instances of objects, and synchronizing a method can decrease performance by a factor of 100.
--> Assuming we would never require more than 1 instance of class that might not be true.
--> Makes testing difficult.

MULTITHREADING SOLUTION

```
/*
NOTE:  once we've set the uniqueInstance variable to an instance of Singleton, we have no further need to synchronize this method. After the first time through, synchronization is totally unneeded overhead!
*/
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
}
```

LAZY CREATED SINGLETON CLASS:

```
/*
Using this approach, we rely on the JVM to create the unique instance of the Singleton when the class is loaded. The JVM guarantees that the instance will be created before any thread accesses the static uniqueInstance variable.
*/
public class Singleton {
	private static Singleton uniqueInstance = new Singleton();

	private Singleton() {
	}

	public static Singleton getInstance() {
		return uniqueInstance;
	}
```

}

<u>“DOUBLE CHECKED LOCKING” to reduce the use of synchronixation in getInstacnce()</u>

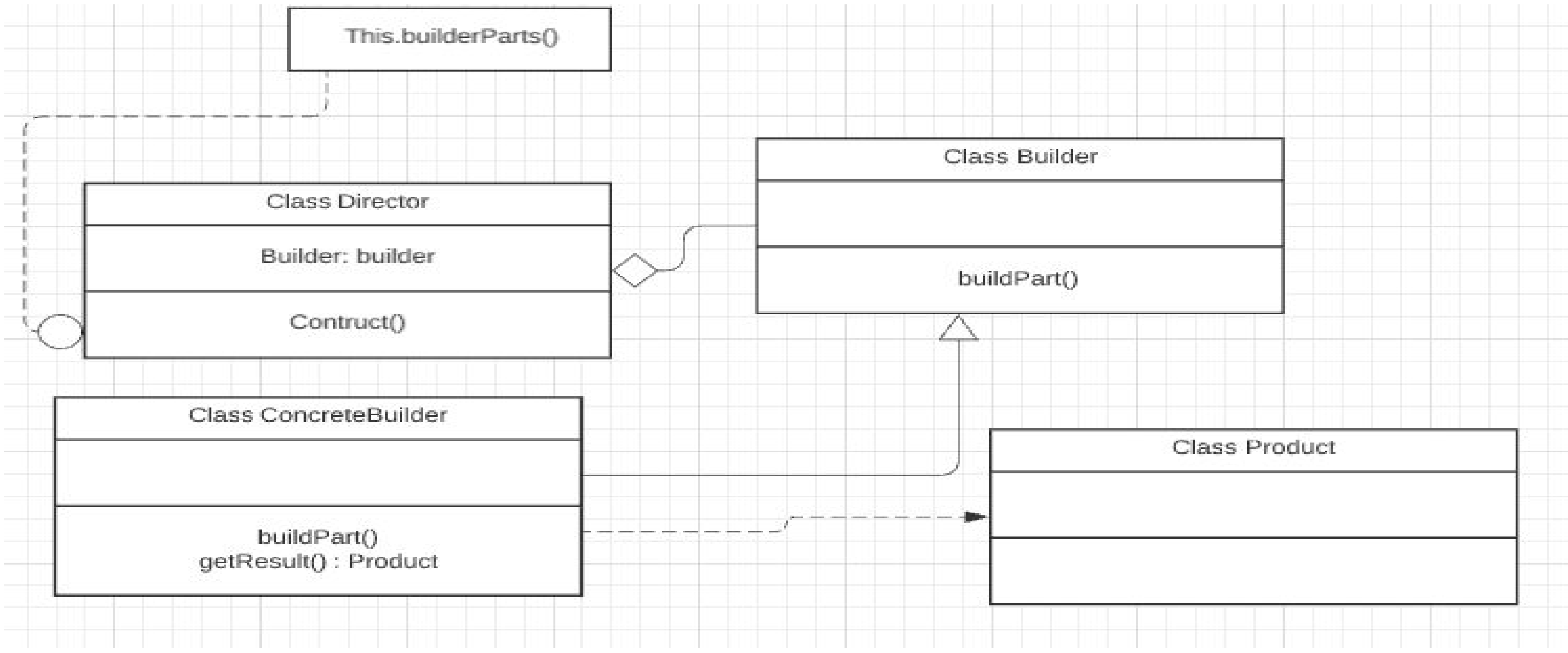
```
/*
With double-checked locking, we first check to see if an instance is created, and if not, THEN we synchronize. This way, we only synchronize the first time through, just what we want.
If performance is an issue in your use of the getInstance() method then this method of implementing the Singleton can drastically reduce the overhead.
*/
public class Singleton {
	private volatile static Singleton uniqueInstance;

	private Singleton() {
	}

	public static Singleton getInstance() {
		if (uniqueInstance == null) {
			synchronized (Singleton.class) {
				if (uniqueInstance == null) {
					uniqueInstance = new Singleton();
				}
			}
		}
		return uniqueInstance;
	}
}
```

4. Design classes for Builder Pattern.

ANS:



This.builderParts()
Class Builder
buildPart()
Class Director
Builder: builder
Contruct()
Class ConcreteBuilder
buildPart()
getResult() : Product
Class Product