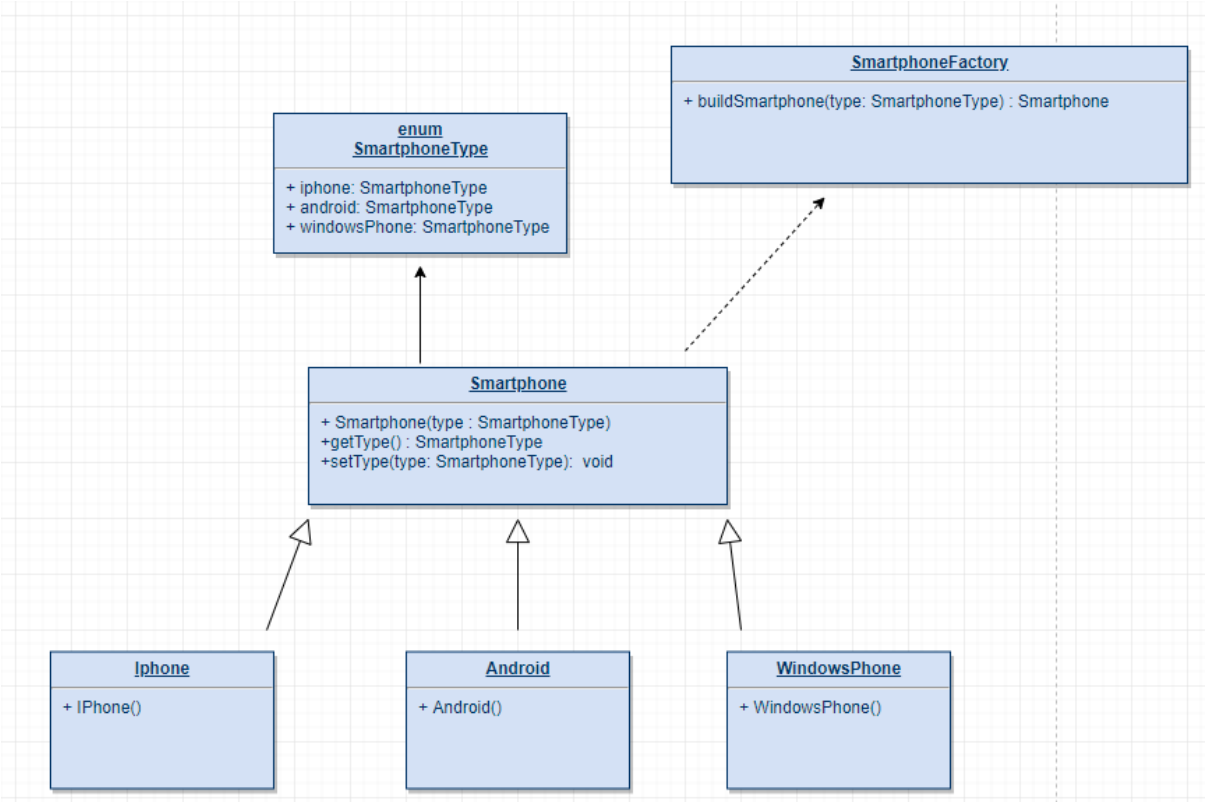
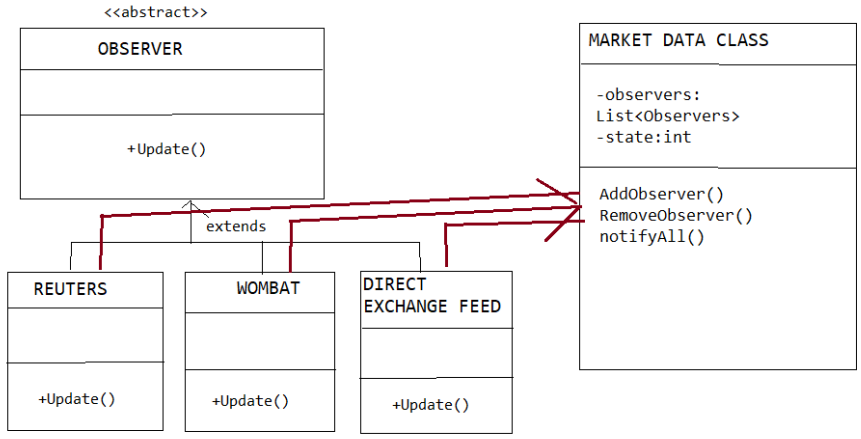
1. You have a Smartphone class and will have derived classes like IPhone, AndroidPhone,WindowsMobilePhone can be even phone names with brand, how would you design this system of Classes.



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 seems the apt design solution to the above problem.

We can create MarketData as an interface or a class, and provide the methods under this class or interface for transactions and obtaining details. Vendors such as Reuter, wombat should be added to the class design via dependency injection principle therefore giving the freedom of changing the vendor from outside the main source code without having to recompile or re deploy. And anybody accessing data will talk directly to MarketData class.



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.

Singleton classes are useful when exactly one object is needed to coordinate actions across the system. By definition it is a software design pattern that restricts the instantiation of a class to one "single" instance.

public class Singleton {

private static Singleton INSTANCE = new Singleton(); private Singleton() {}

public static Singleton getInstance() {

return INSTANCE;

}

}

Thread Safe Singleton

To create a thread-safe singleton class, we can have a global access method synchronized which means only thread will get access. Example code :

synchronize implementation -

```
public class ThreadSafeSingleton {

private static ThreadSafeSingleton instance;
private ThreadSafeSingleton(){}

public static synchronized ThreadSafeSingleton getInstance(){ if(instance == null){

instance = new ThreadSafeSingleton();
}
return instance;
}

}
```

Lazy Created Singleton Class:

JVM creates 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;

}

}
```

Different ways of handling multiple singleton classes are:

1. Reflection:

Reflection can be caused to destroy singleton property of singleton class. To overcome issue raised by reflection, enums are used because java ensures internally that enum value is instantiated only once. Since java Enums are globally accessible, they can be used for singletons.

```
public enum ReflectionSingleton {

INSTANCE;
public static void logicCode(){    }
}
```

2. Serialization:

Serialization is a technique which causes breakage of singleton property of singleton classes. If we serialize an object of a singleton class and then de-serialize that object it will create a new instance completely and hence breaks the singleton pattern.

Implementation of readResolve() method.

```
protected Object readResolve() {
```

return getInstance();

}

3. Cloning -

Clone can create copy of object. If we create clone of a singleton object, there will be two instances of a singleton class, breaking singleton property.

We can solve this problem by overriding clone() method and throw an exception from clone method that is CloneNotSupportedException

@Override

protected Object clone() throws CloneNotSupportedException

{

return instance;

}

4. Processor is built in hierarchy to demonstrate Builder Pattern