Design Principles and Patterns By Mohit Gupta

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

Question-1) 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

Solution-

Strategy Pattern can be used to solve the above question.

STEPS-

1) Create a Parent Class "SmartPhone"

2) It can have SmartPhone must have attributes like Model,Release Year, Brand.

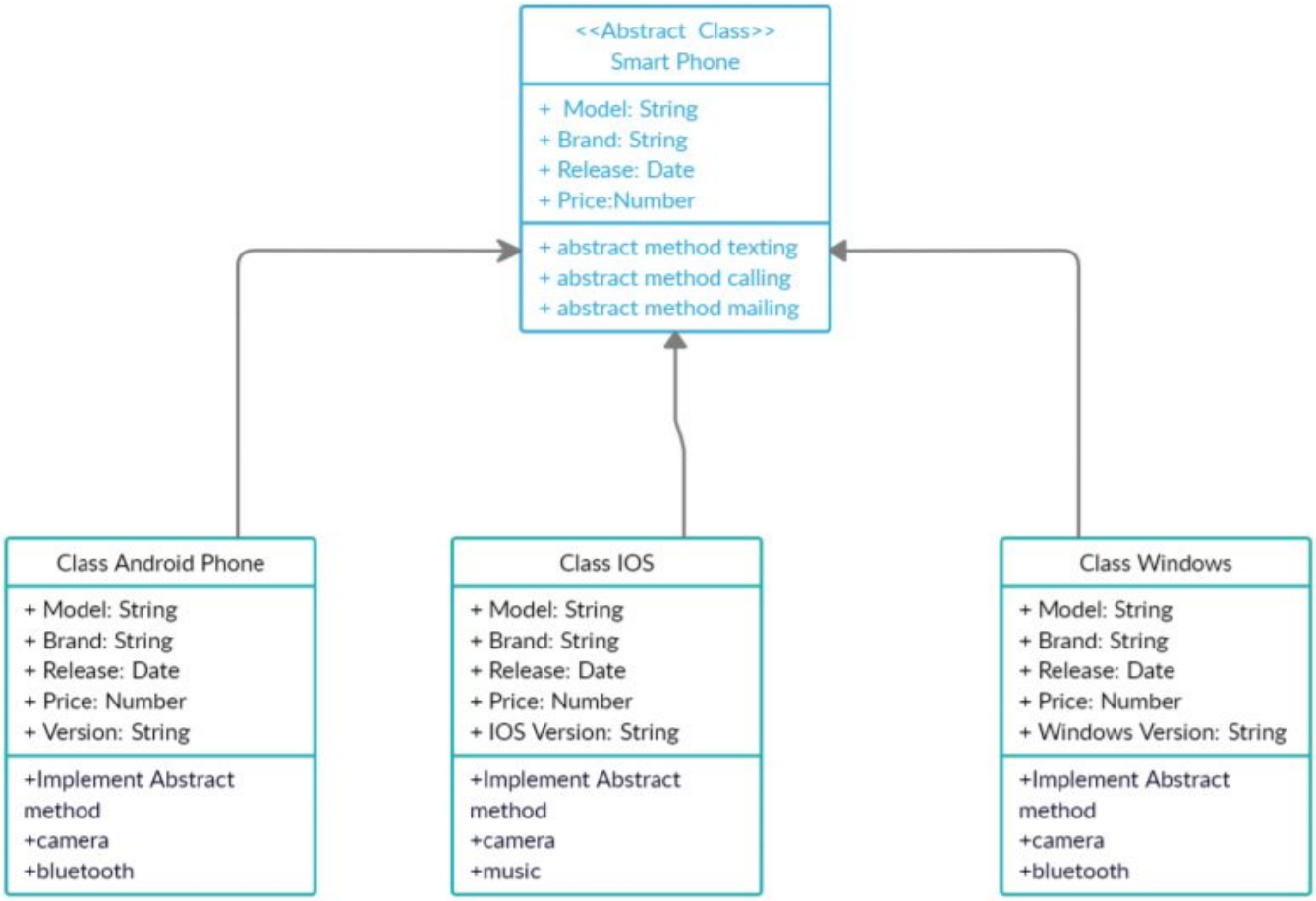
3) Parent class should also include Smartphone must have features like Calling,Messaging as abstract methods

4) For Common SmartPhone features,You can have Interfaces that Indicates SmartPhone common features like Camera,Player etc

5) Then Child Classes like AndroidPhone ,iOSPhone can inherit the SmartPhone Parent Class and shall define all must have attributes and features of a SmartPhone.

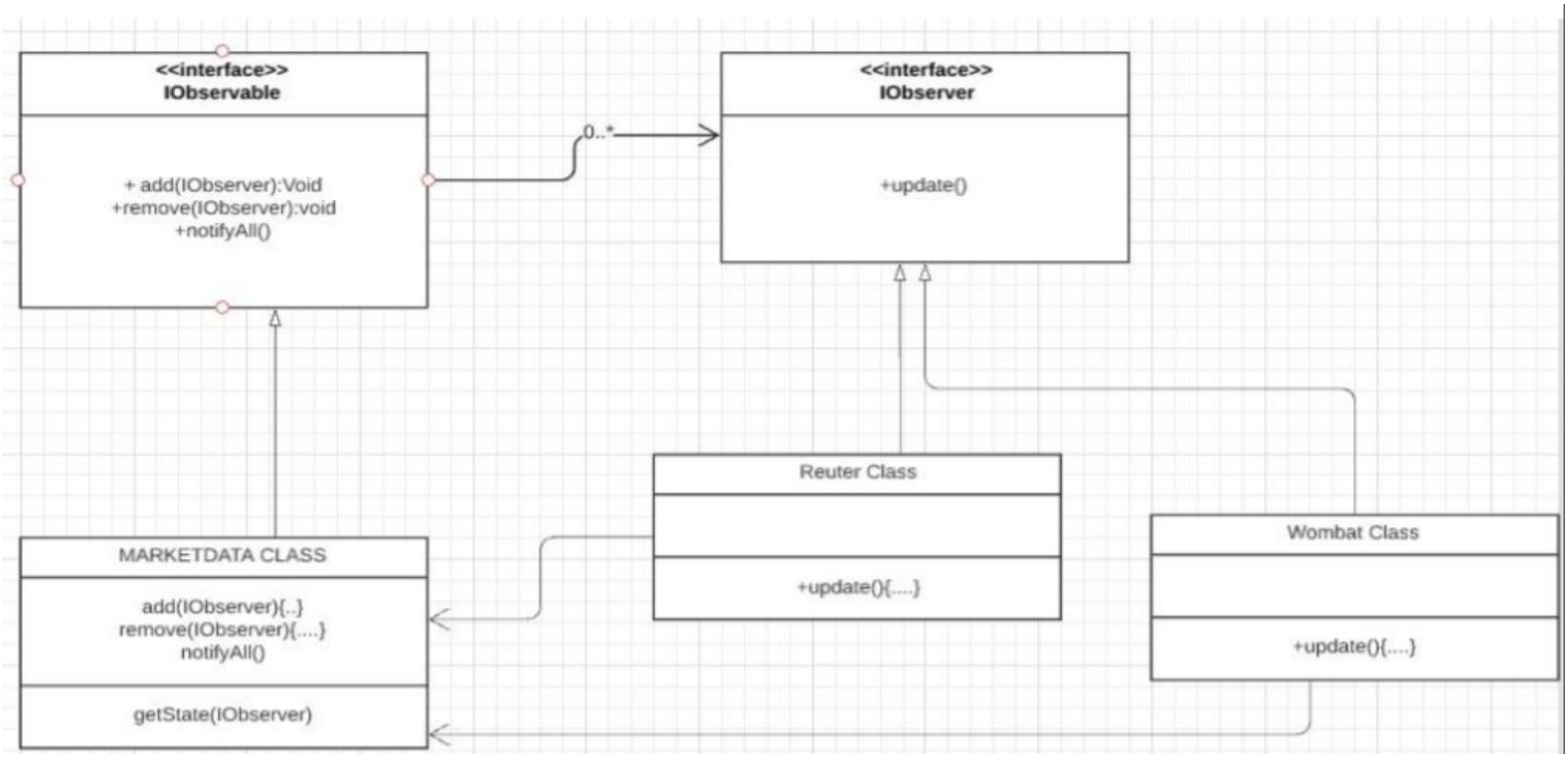
6) Child Classes then can implement common features Interfaces , if those features are supported by that Phone.

Diagram-


<<Abstract Class>>
Smart Phone
+ Model: String
+ Brand: String
+ Release: Date
+ Price:Number
+ abstract method texting
+ abstract method calling
+ abstract method mailing
Class Android Phone
+ Model: String
+ Brand: String
+ Release: Date
+ Price: Number
+ Version: String
+Implement Abstract method
+camera
+bluetooth
Class IOS
+ Model: String
+ Brand: String
+ Release: Date
+ Price: Number
+ IOS Version: String
+Implement Abstract method
+camera
+music
Class Windows
+ Model: String
+ Brand: String
+ Release: Date
+ Price: Number
+ Windows Version: String
+Implement Abstract method
+camera
+bluetooth

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

Solution- The Observer Design Pattern can be used for the above.



Question-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-

DEFINITION - Singleton Pattern says that just "define a class that has only one instance and provides a global point of access to it".

USE CASE- Common use-cases for the singleton design pattern include factories, builders, and objects that hold program state.

ADVANTAGES-

1) Singletons can implement interfaces
2) Singletons can be passed as parameters
3) Singletons can have their instances swapped out (such as for testing purposes)
4) Singletons can be handled polymorphically, so there may exist multiple implementations

Thread Safe Singleton-

The version of lazy initialization shown above is not thread-safe.Additionally, this can result in threads receiving a partially-created singleton object

The below implementation is thread-safe through the use of synchronization. The instance variable is now also declared as volatile , which ensures that all threads have an updated view of the singleton instance.

```
public final class Singleton {

   private static volatile Singleton instance = null;

   private Singleton() {}

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

      return instance;
   }
}
```

Multiple Classes-

```
public class Singleton {
    private Singleton() {}

    private static class LazyHolder {
        private static final Singleton INSTANCE = new Singleton();
    }

    public static Singleton getInstance() {
        return LazyHolder.INSTANCE;
    }
}
```

Question-4- Design classes for Builder Pattern.

Solution- Builder pattern builds a complex object using simple objects and using a step by step approach. This type of design pattern comes under the creational pattern as this pattern provides one of the best ways to create an object.

A Builder class builds the final object step by step. This builder is independent of other objects.