T.VIJAYA SREE

SAU Batch-1

DESIGN PRINCIPLE AND PATTERN

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.

Ans:

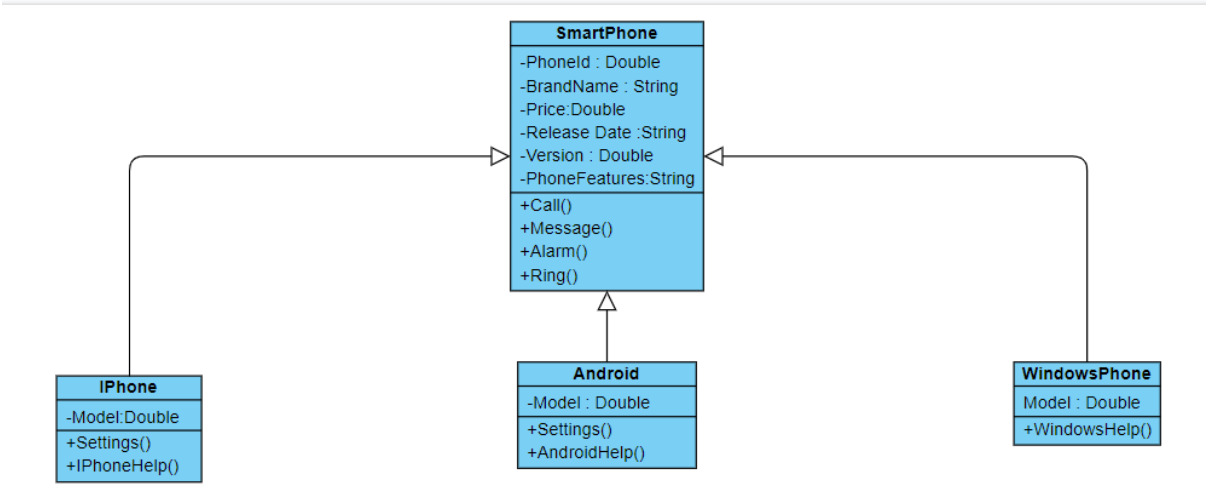
→There will be a parent abstract class called Smart Phone having member attributes PhoneId,BrandName,Price,PhoneFeatures,Version,ReleaseDate

-> SmartPhone will have member functions like call(),alarm(),ring(),message()

-> iPhone, Android and Windows are child classes derived from the Smart Phone class.

-> We have a Smartphone factory that gives the appropriate phone based on the choice selected.

-> The abstract class has one abstract method of find the name based on the phone selected.



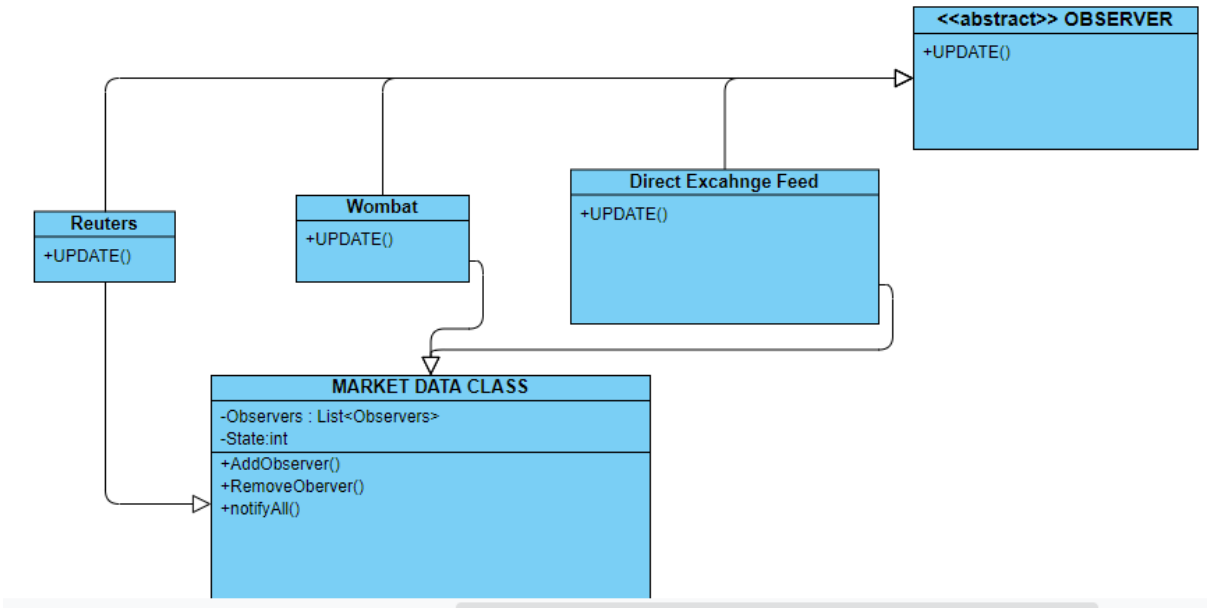
```
public public class SmartPhone {
    double phoneId;
    String brandName;
    double price;
    String ReleaseDate;
    String PhoneFeatures;
    double version;
    public void call(){
        System.out.println("Calling");
    }
    public void message(){
        System.out.println("Messaging");
    }
    public void alarm(){
        System.out.println("Alarm");
    }
    public void ring(){
        System.out.println("Mobile is Rin
ging");
    }
}



public class Android{
     double model;
     String AndroidHelp(){
```

```
        System.out.println("Android help
");
    }
    String AndroidSettings(){
        System.out.println("Android Sett
ings");
    }
}


public class Iphone{
    double model;
    String IphoneHelp(){
        System.out.println("Iphone help")
;
    }
    String IphoneSettings(){
        System.out.println("Iphone Settin
gs");
    }
}

public class WindowsPhone{
    double model;
    String WindowsHelp(){
        System.out.println("WindowsPhone
help");
    }
    String WindowsSettings(){
```

```
        System.out.println("WindowsPhone
Settings");
    }
}
```

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.

   Ans: We can use the Observer Design Pattern for the above scenario because one-to-many relationship between objects can be done.



```
public abstract class Observer
{
    public abstract void update();
}


public class MarketData{
    private List<Observer> observer = new
 ArrayList<>();
    private int state;
    public int getState()
    {
        return state;
    }
    public  setState(int state)
    {
        this.state=state;
        notifyAllObservers();
    }
    public void addObserver(Observer obse
rver)
    {
        observer.add(observer);
    }

    public void removeObserver(Observer o
bserver)
    {
```

```
        observer.remove(observer);
    }
}

public class Reuters extends Observer
{
    @override
    public void update()
    {
        System.out.println("Retuers updat
ing");
    }
}
public class Wombat extends Observer
{
    @override
    public void update()
    {
        System.out.println("Wombat updati
ng");
    }
}
public class DirectFeed extends Observer
{
    @override
    public void update()
    {
        System.out.println("DirectFeed up
dating");
```

```
        }
}
```

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:

This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.
Eager initialization: This is the simplest method of creating a singleton class. In this, object of class is created when it is loaded to the memory by JVM. It is done by assigning the reference an instance directly.

```
// Eager Initialization
public class singleton
{
 // public instance initialized when loading the class
 private static final GFG instance = new singleton ();

 private singleton ()
 {
  // private constructor
```

```
  }
  public static singleton getInstance(){
       return instance;
    }
}
```

Lazy initialization: In this method, object is created only if it is needed. This may prevent resource wastage.

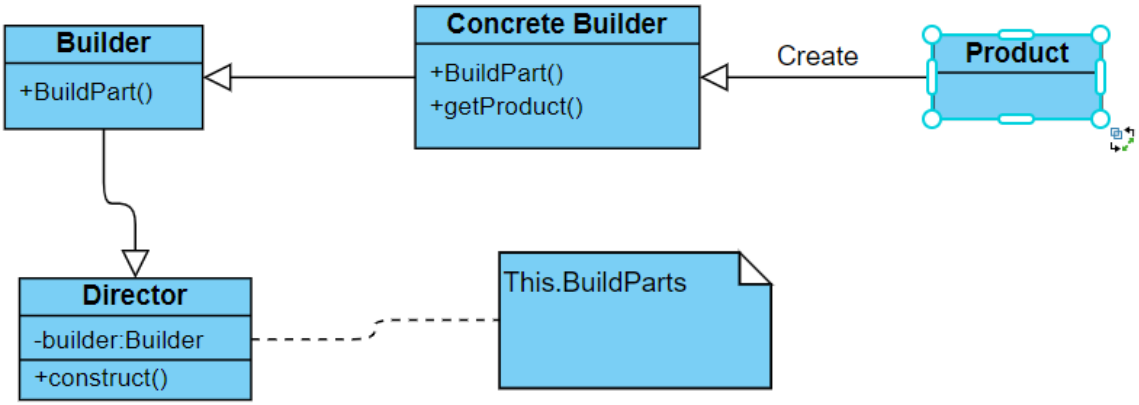
```
// With Lazy initialization
public class singleton
{
  // private instance, so that it can be
  // accessed by only by getInstance() method
  private static singleton instance;

  private singleton ()
  {
    // private constructor
  }

  //method to return instance of class
  public static singleton getInstance()
  {
    if (instance == null)
    {
      // if instance is null, initialize
      instance = new singleton ();
    }
    return instance;
  }
}
```

Thread Safe Singleton: A thread safe singleton in created so that singleton property is maintained even in multithreaded environment. To make a singleton class thread-safe, getInstance() method is made synchronized so that multiple threads can’t access it simultaneously.

```
// Java program to create Thread Safe
// Singleton class
public class Java
{
  // private instance, so that it can be
  // accessed by only by getInstance() method
  private static Java instance;

  private Java()
  {
    // private constructor
  }

 //synchronized method to control simultaneous access
  synchronized public static Java getInstance()
  {
    if (instance == null)
    {
      // if instance is null, initialize
      instance = new Java();
    }
    return instance;
  }
}
```

4. 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.


Builder
+BuildPart()
Concrete Builder
+BuildPart()
+getProduct()
Create
Product
Director
-builder:Builder
+construct()
This.BuildParts