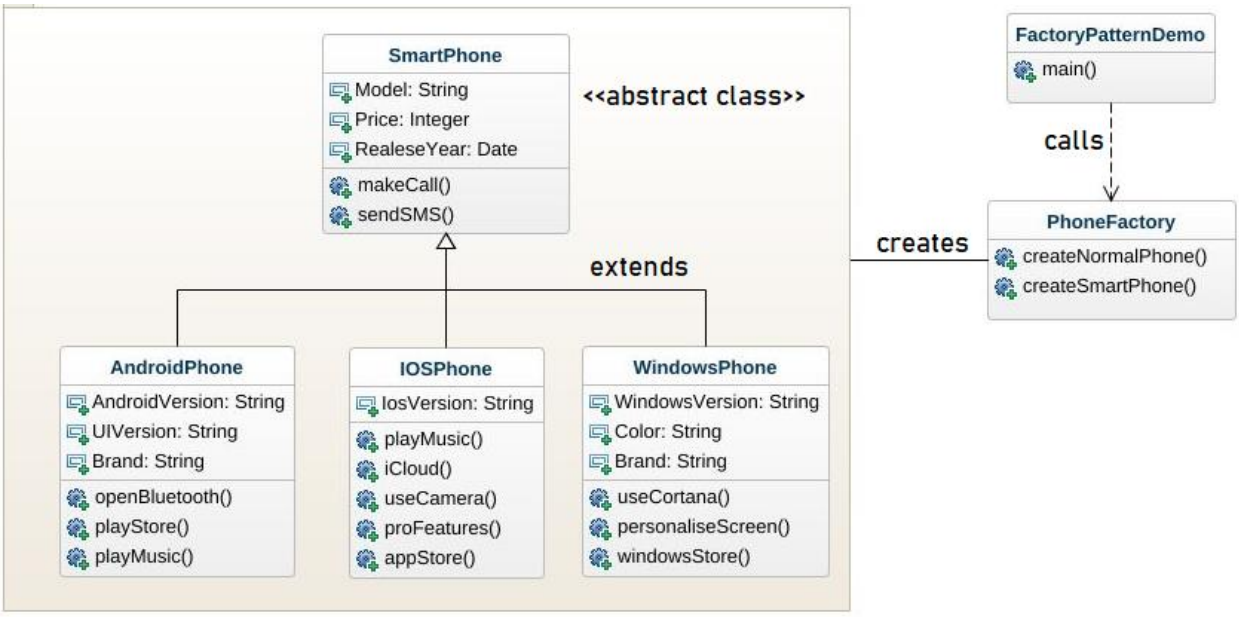
Accolite University Feb Batch 1
Design Principles and Patterns Assignment

- Meet Shah
(INT626)

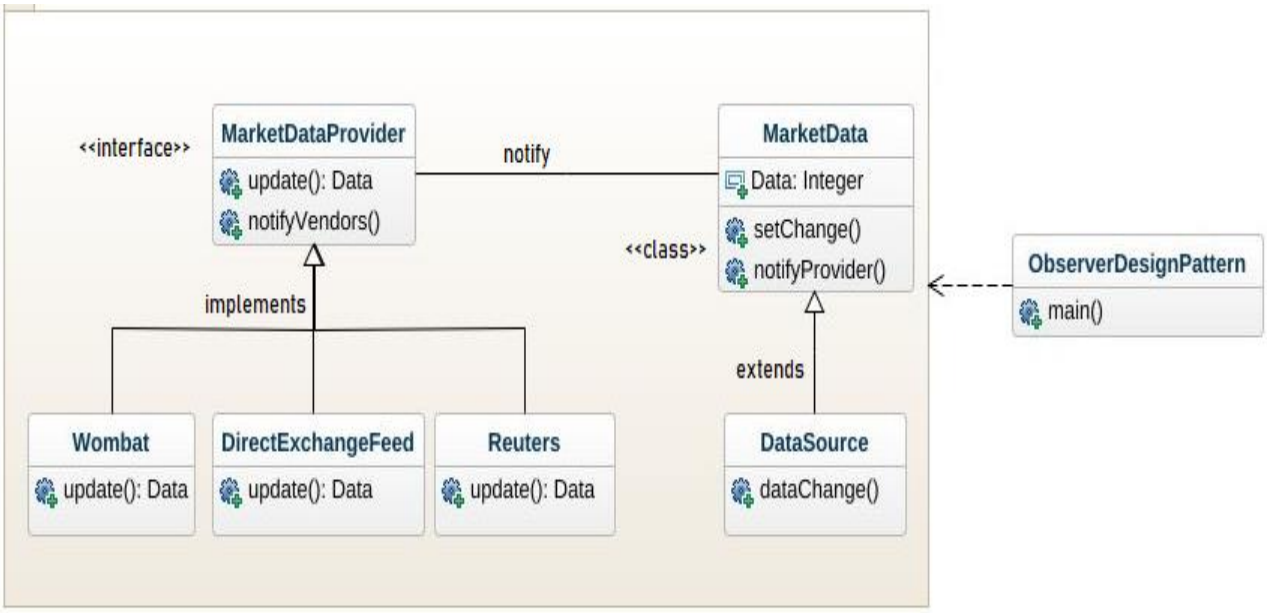
1. **You have a Smartphone class and will have derived classes like IPhone, Android Phone, Windows Mobile Phone can be even phone names with brand and how would you design this system of classes.**

   Factory Pattern can be used for the given scenario. We can define an interface or abstract class for creating an object and let the subclasses decide which class to instantiate.



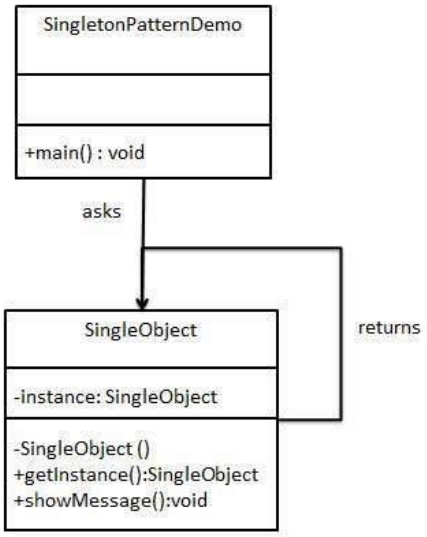
2. **Write classes to provide Market Data and you know that you can switch to different vendors overtime like Reuters, wombat and may be even to direct exchange feed, how do you design your Market Data system.**

   Observer Pattern can be used for the given scenario. It says that "just define a one-to-one dependency so that when one object changes state, all its dependents are notified and updated automatically". Any update in the price stocks will be automatically pushed to all its vendors.



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 Pattern says that just "define a class that has only one instance and provides a global point of access to it". In other words, a class must ensure that only single instance should be created and single object can be used by all other classes.



There are two forms of singleton design pattern:

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

Advantage:

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

Usage:

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

To create the singleton class, we need to have static member of class, private constructor and static factory method.

- Static member: It gets memory only once because of static, it contains the instance of the Singleton class.

- Private constructor: It will prevent to instantiate the Singleton class from outside the class.
- Static factory method: This provides the global point of access to the Singleton object and returns the instance to the caller.

There are many ways this can be done in Java. All these ways differs in their implementation of the pattern, but in the end, they all achieve the same end result of a single instance.

1. Early Initialisation: 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.

```
public class Demo
{
  private static final Demo instance = new Demo();

  private Demo()
  {
  }

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

2. Lazy Initialisation: In this method, object is created only if it is needed. This may prevent resource wastage. An implementation of getInstance() method is required which return the instance.

```
public class Demo
{
  private static Demo instance;

  private Demo()
  {
  }

  public static Demo getInstance()
  {
    if (instance == null)
    {
      instance = new Demo();
    }
    return instance;
  }
}
```

3. 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.

```
public class Demo
{
  private static Demo instance;

  private Demo()
  {
  }

  synchronized public static Demo getInstance()
  {
    if (instance == null)
    {
      instance = new Demo();
    }
    return instance;
  }
}
```

4. Lazy Initialisation with Double check locking: In this mechanism, we overcome the overhead problem of synchronized code. In this method, getInstance() is not synchronized but the block which creates instance is synchronized so that minimum number of threads have to wait and that's only for first time.

```
public class Demo
{
  private static Demo instance;

  private Demo(){
  }

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

      }
    }
    return instance;
  }
}
```

4. Design classes for Builder Pattern.

The builder pattern is a design pattern that allows for the step-by-step creation of complex objects using the correct sequence of actions. The construction is controlled by a director object that only needs to know the type of object it is to create.

In Builder pattern, we have a inner static class named Builder inside our Server class with instance fields for that class and also have a factory method to return an new instance of Builder class on every invocation. The setter methods will now return Builder class reference.