Singleton

```
class Main {
   public static void main(String[] args) {
   //   Singleton obj1 = Singleton.getInstance();
   //   Singleton obj2 = Singleton.getInstance();
   Thread t1 = new Thread(new Runnable()
   {
     public void run(){
         Singleton obj1 = Singleton.getInstance();
     }
   });

   Thread t2 = new Thread(new Runnable()
   {
     public void run(){
         Singleton obj1 = Singleton.getInstance();
     }
   });
   t1.start();
   t2.start();
   }
}

class Singleton {
   // Early
   // static Singleton instance = new Singleton();
   static Singleton instance;
   private Singleton(){
       System.out.println("Instance created..");
   }

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

Strategy

```
interface Strategy {
   public int performOperation(int a, int b);
}

class Addition implements Strategy {
   public int performOperation(int a, int b) {
      return a + b;
   }
}
class Multiplication implements Strategy {
   public int performOperation(int a, int b) {
      return a * b;
   }
}
class Subtraction implements Strategy {
   public int performOperation(int a, int b) {
      return a - b;
   }
}

class Context {
   private Strategy strategy;

   public Context(Strategy strategy){
      this.strategy = strategy;
   }

   public int doStrategy(int a, int b){
      return strategy.performOperation(a, b);
   }
}

class mainClass {
    public static void main(String[] args) {
      Context context = new Context(new Addition());
      System.out.println("Addition: " + context.doStrategy(30, 20));

      context = new Context(new Multiplication());
      System.out.println("Multiplication: " + context.doStrategy(30,
20));
    }
}
```

Sorting Strategy

```java
interface Sorting {
   public String performOperation(int[] args);
}

class BubbleSort implements Sorting {
   public String performOperation(int[] args) {
      return "BubbleSort is Done";
   }
}

class MergeSort implements Sorting {
   public String performOperation(int[] args) {
      return "MergeSort is Done";
   }
}

class QuickSort implements Sorting {
   public String performOperation(int[] args) {
      return "QuickSort is Done";
   }
}

class Context {
   private Sorting sorting;

   public Context(Sorting sorting){
      this.sorting = sorting;
   }

   public String doStrategy(int[] args){
      return sorting.performOperation(args);
   }
}

class Owner {
    public static void main(String[] args) {
      int[] array = {5, 3, 8, 4, 2};

      Context context = new Context(new BubbleSort());
      System.out.println("BubbleSort: " + context.doStrategy(array));

      context = new Context(new MergeSort());
      System.out.println("MergeSort: " + context.doStrategy(array));

      context = new Context(new QuickSort());
      System.out.println("QuickSort: " + context.doStrategy(array));
    }
}
```

Decorator

```java
interface Coffee {
    String getDescription();
    double getCost();
}
class PlainCoffee implements Coffee {
    public String getDescription() {
        return "Plain Coffee";
    }
    public double getCost() {
        return 200;
    }
}
class CoffeeDecorator implements Coffee {
    protected Coffee decoratedCoffee;
    public CoffeeDecorator(Coffee decoratedCoffee) {
        this.decoratedCoffee = decoratedCoffee;
    }
    public String getDescription() {
        return decoratedCoffee.getDescription();
    }
    public double getCost() {
        return decoratedCoffee.getCost();
    }
}

class MilkDecorator extends CoffeeDecorator {
    public MilkDecorator(Coffee decoratedCoffee) {
        super(decoratedCoffee);
    }
    public String getDescription() {
        return decoratedCoffee.getDescription() + ", Milk";
    }
    public double getCost() {
        return decoratedCoffee.getCost() + 100;
    }
}

class SugarDecorator extends CoffeeDecorator {
    public SugarDecorator(Coffee decoratedCoffee) {
        super(decoratedCoffee);
    }
    public String getDescription() {
        return decoratedCoffee.getDescription() + ", Sugar";
    }
    public double getCost() {
        return decoratedCoffee.getCost() + 50;
    }
}

public class MainClass {
    public static void main(String[] args) {
        // Plain Coffee
        Coffee coffee = new PlainCoffee();
        System.out.println("Description: " +
coffee.getDescription());
        System.out.println("Cost: BDT " + coffee.getCost());

        // Coffee with Milk
        Coffee milkCoffee = new MilkDecorator(new PlainCoffee());
        System.out.println("\nDescription: " + milkCoffee.getDescription());
        System.out.println("Cost: BDT " + milkCoffee.getCost());

        // Coffee with Sugar and Milk
        Coffee sugarMilkCoffee = new SugarDecorator(new MilkDecorator(new PlainCoffee()));
        System.out.println("\nDescription: " + sugarMilkCoffee.getDescription());
        System.out.println("Cost: BDT " + sugarMilkCoffee.getCost());
    }
}
```

Factory

```
interface OS{
    String show();
}
class Android implements OS{
    public String show(){
        return "This is android.";
    }
}
class IOS implements OS{
    public String show(){
        return "This is IOS.";
    }
}
interface Maker{
    OS makeOS();
}
class AndroidMaker implements Maker{
    public OS makeOS(){
        return new Android();
    }
}
class IOSMaker implements Maker{
    public OS makeOS(){
        return new IOS();
    }
}
class Factory{
    public OS getOS(String name){
        Maker maker;
        if(name == "Android"){
            maker = new AndroidMaker();
        }
        else{
            maker = new IOSMaker();
         }

        return maker.makeOS();
    }
}
public class Main {
    public static void main(String[] args) {
        Factory factory = new Factory();
        OS os = factory.getOS("Android");
        System.out.println(os.show());
    }
}
```

Command

```
import java.util.ArrayList;
import java.util.List;

interface Order {
   void execute();
}

class BuyStock implements Order {
   private Stock abcStock;
   public BuyStock(Stock abcStock){
      this.abcStock = abcStock;
   }
   public void execute() {
      abcStock.buy();
   }
}

class SellStock implements Order {
   private Stock abcStock;
   public SellStock(Stock abcStock){
      this.abcStock = abcStock;
   }
   public void execute() {
      abcStock.sell();
   }
}

class Stock {
   private String name = "ABC";
   private int quantity = 10;
   public void buy(){
      System.out.println("Stock [ Name: " +name+", Quantity: " +
quantity +" ] bought");
   }
   public void sell(){
      System.out.println("Stock [ Name: "+name+", Quantity: " +
quantity +" ] sold");
   }
}
```

```
class Broker {
   private List<Order> orderList = new ArrayList<Order>();
   public void takeOrder(Order order){
      orderList.add(order);
   }
   public void placeOrders(){
      for (Order order : orderList) {
         order.execute();
      }
      orderList.clear();
   }
}


public class CommandPatternDemo {
   public static void main(String[] args) {
      Stock abcStock = new Stock();

      BuyStock buyStockOrder = new BuyStock(abcStock);
      SellStock sellStockOrder = new SellStock(abcStock);

      Broker broker = new Broker();
      broker.takeOrder(buyStockOrder);
      broker.takeOrder(sellStockOrder);

      broker.placeOrders();
   }
}
```