* Factory Design Pattern in Java:

. Factory Main.Java

```
public interface OS {
  public void spec();
}
class Android implements OS {
  public void spec() {
     SOP ("Most Powerful OS");
  }
}
class IOS implements OS {
  @override
  public void spec() {
     SOP ("Most Secure OS");
  }
}
class Windows implements OS {
  @Override
  public void spec() {
    SOP ("I am about to die...");
  }
}
class Factory Main {
  psvm (String [] args) {
     OS obj = new Android();
     OS obj = new IOS();
     OS obj = new Windows();
     obj.spec();
  }
}
```

- Factory Main1.java

```
public interface OS {
  public void spec();
}
class Andriod implements OS {
  public void spec() {
    SOP ("Most powerful OS");
  }
}
class IOS implements OS {
  @Override
  public void spec() {
    SOP ("Most secure OS");
  }
}
class Windows implements OS {
  @Override
  public void spec() {
    SOP ("I am about to die...");
  }
}
class Operating system Factory {
   OS getInstance (String str) {
    if (str.equals ("open"))
      return new Android();
    else if (str.equals ("closed"))
      return new IOS();
    else
      return new Windows();
  }
}
```

```
class FactoryMain1 {
  psvm(String args[]) {
   OperatingSystemFactory osf = new OperatingSystemFactory();
   OS obj = osf.getInstance("open");
   obj.spec();                    // "closed"
  }                               // "idkjf"
}
```

* Builder Design Pattern in Java:

- Phone.java

```
package com.phone ; // Builder Pattern
public class Phone {
 private string OS;
 private int ram;
 private string processor;
 private double screensize;
 private int battery;
 public Phone (String OS, int ram, string processor,
               double screensize, int battery){
  super();
  this.os = os;
  this.ram = ram;
  this.processor = processor;
  this.screensize = screensize;
  this.battery = battery;
}
@override
public string tostring() {
  return "phone [os=" +ost", ram=" + ram t", processor=
  " + processort", screensize=" + screensize t", battery =
  " +battery"] ";
}
```

shop.java

```
public class shop {
  psvm (String[] args) {
    Phone p = new Phone ("Android", 2, "Qualcomm",
                                        5.5, 3100);

    sop(p);
  }
}
```

- New Program Creational / Design Pattern :

PhoneBuilder.java

```
public class PhoneBuilder {
  private string os;
  private int ram;
  private string processor;
  private double screensize;
  private int battery;

  public PhoneBuilder setOs (String os) {
    this.os = os;
    return this;
  }
  public PhoneBuilder setRam (int ram) {
    this.ram = ram;
    return this;
  }
  public PhoneBuilder Processor (String processor) {
    this.processor = processor;
    return this;
  }
  public PhoneBuilder setScreenSize (double screensize) {
```

```
    this.screensize = screensize;
    return this;
  }
  public PhoneBuilder setBattery(int battery){
    this.battery = battery;
    return this;
  }
  public Phone getPhone(){
    return new Phone(os, ram, processor, screensize,
                                          battery);
  }
}
```

shop.java

```
public class Shop{
  psvm(String[] args){
    Phone p = new PhoneBuilder().setOs("Android").
              setRam(2).getPhone();
    SOP(p);
  }
}
```

* Adapter Design Pattern in Java:

- Pen.java

```
public interface Pen{
  void write(String str);
}
```

- PilotPen.java{

```
  public void mark(String str){
    SOP(str);
  }
}
```

- PenAdapter.java

```
public class PenAdapter implements Pen {
  pilot_Pen pp = new PilotPen();
  @Override
  public void write (String str) {
    pp.mark (str);
  }
}
```

- AssignmentWork.java

```
public class AssignmentWork{
  private Pen p;
  public Pen getP() {
    return p;
  }
  public void setP (Pen p) {
    this.p = p;
  }
  Public void writeAssignment (String str) {
    p.write (str);
  }
}
```

- School.java

```
public class school {
  psvm (String [] args) {
    PilotPen pp = new PilotPen();
    Pen p = new PenAdapter ();
    AssignmentWork aw = new AssignmentWork ();
    aw.setP(p);
    aw.writeAssignment ("I'm bit tired to write assignment");
  }
}
```

* Composite Design Pattern in Java :

. CompositeTest.java

```
public class CompositeTest {
  psvm (String[] args) {
    Component hd = new Leaf (4000, "Hpp");
    Component mouse = new Leaf (500, "Mouse");
    Component monitor = new Leaf (8000, "Monitor");
    Component ram = new Leaf (3000, "Ram");
    Component cpu = new Leaf (9000, "CPU");

    Composite ph = new Composite ("Peri");
    Composite cabinet = new Composite ("Cabinet");
    Composite mb = new Composite ("MB");
    Composite computer = new Composite ("Computer");

    mb.addComponent (cpu);
    mb.addComponent (ram);
    ph.addComponent (mouse);
    ph.addComponent (monitor);
    cabinet.addComponet (hd);
    cabinet.addComponent (mb);

    Computer.addComponent (ph);
    Computer.addComponent (cabinet);
    // Ph.showPrice ();
    // Computer.showPrice ();
    ram.showPrice ();
  }
}
```

Component.java

```
import java.util.ArrayList;
import java.util.List;
interface component {
    void showPrice();
}
class leaf implements Component {
    int price;
    string name;
    public leaf (int price, string name) {
        super();
        this.price = price;
        this.name=name;
}
@override
public void showPrice() {
    SOP ("name+" :" + price);
}
class Composite implements Component {
    string name;
    List<Component> components = new ArrayList<>();
    public Composite (string name) {
        super();
        this.name = name;
    }
    public void addComponent (Component com) {
        Components.add(com);
}
@Override
public void showPrice() {
    SOP (name);
```

```
        for (Component c : components)
        {
            c.showPrice();
        }
    }
}
```

* Prototype Design Pattern in Java :

- Demo.java

```
public class Demo{
 psvm(String[] args) throws CloneNotSupportedException {
   Bookshop bs = new Bookshop();
   bs.setShopName("Novelty");
   bs.loadData();

   Bookshop bs1 = bs.clone();
   bs.getBooks().remove(2);
   bs1.setShopName("A1");
   SOP(bs);
   SOP(bs1);
 }
}
```

- Book.java

```
public class Book{
 private int bid;
 private String bname;
 public int getBid(){
  return bid;
 }
```

```
    public void setBid(int bid){
      this.bid = bid;
    }
    public string getBname(){
      return bname;
    }
    public void setBname (String bname){
      this.bname = bname;
    }
    @Override
    public string toString(){
      return "Book [bid =" + bid + ", bname = " + bname + "]";
    }
}
```

- BookShop.java

```
import java.util.ArrayList;
import java.util.List;
class Bookshop implements cloneable {
  private string shopName;
  list<Book> books = new ArrayList<>();
  public string getShopName(){
    return ShopName;
  }
  public void setshopName (string shopName){
    this.shopName = shopName;
  }
  public Lisk <Book> getBooks(){
    return books;
  }
  public void setBooks (List <Book> books) {
```

```
        this.books = books;
    }
    public void loadData () {
        for (int i = 1; i <= 10; i++)
        {
            Book b = new Book();
            b.setBid (i);
            b.setBname ("Book" + i);
            getBooks().add(b);
        }
    }
    @Override
    public String toString () {
        return "Bookshop [shopName =" + shopName + ", books ="
                                        + books + "]";
    }
    @Override
    protected Bookshop clone() throws CloneNotSupported-
                                        Exception {
        Bookshop shop = new Bookshop();
        for (Book b : getBooks()) {
            shop.getBooks().add(b);
        }
        return shop;
    }
}
```

* Observer Design Pattern in Java :

- Youtube.java

```
public class Youtube {
```

```
psvm (String [] args) {
 Channel telusko = new channel();
 subscriber s1 = new subscriber ("Akshay");
 subscriber s2 = new subscriber ("soham");
 subscriber s3 = new subscriber ("Harsh");
 subscriber s4 = new subscriber ("kiran");
 subscriber s5 = new subscriber ("Pravin");

 telusko.subscriber (s1);
 telusko.subscriber(s2);
 telusko.subscriber(s3);
 telusko.subscriber(s4);
 telusko.subscriber(s5);

 telusko.unsubscriber(s3);

 s1.subscriberChannel (telusko);
 s2.subscriberChannel (telusko);
 s3.subscribeChannel (telusko);
 s4.subscribeChannel (telusko);
 s5.subscribeChannel (telusko);

 telusko.upload ("How to learn Programming ??");
 }
}
```

- subscriber.java

```
public class subscriber {
 private string name;
 private channel channel = new Channel();
 public subscriber (string name) {}
```

```
    super();
    this.name = name;
  }
  public void update() {
   SOP("Hey" + name + ", Video Uploaded:" + channel.title);
  }
  public void subscribeChannel (channel ch) {
    channel = ch;
  }
}
```

- Channel.java

```
import java.util.ArrayList;
import java.util.List;
public class channel {
  private List<Subscriber> subs = new ArrayList<>();
  string title;
  public void subscribe(Subscriber sub) {
    subs.add(sub);
  }
  public void unSubscribe(SUbscriber sub) {
   subs.remove(sub);
  }
  public void notifySubscribers() {
   for (Subscriber sub: subs) {
    sub.update();
   }
}
}
public void upload (string title) {
  this.title = title;
  notifySubscribers();
}
}
```

:)