Exercise 1: Implementing the Singleton Pattern

code:

class Logger {

    private static Logger obj=null;

    private Logger(){}

    public synchronized static Logger get(){

        if(obj==null) obj=new Logger();

        return obj;

    }

    public void ad(){

        System.out.println("Address of the object is :"+obj);

    }

    public static void main(String[] args) {

        Logger x=Logger.get();

        Logger y=Logger.get();

        x.ad();

        y.ad();

    }

}

output:



Exercise 2: Implementing the Factory Method Pattern

Code:

abstract class Doc{

    public abstract void readDoc();

}

class ConcreteWordDoc extends Doc{

    public void readDoc(){

        System.out.println("Reading Word Document");

    }

}

class ConcretePDFDoc extends Doc{

    public void readDoc(){

        System.out.println("Reading PDF Document");

    }

}

class ConcreteExcelDoc extends Doc{

    public void readDoc(){

        System.out.println("Reading Excel Document");

    }

}

abstract class DocFactory{

    abstract Doc createDoc();

}

class WordDocFactory extends DocFactory{

    Doc createDoc(){

        return new ConcreteWordDoc();

    }

}

class PDFDocFactory extends DocFactory{

    Doc createDoc(){

        return new ConcretePDFDoc();

    }

}

class ExcelDocFactory extends DocFactory{

    Doc createDoc(){

        return new ConcreteExcelDoc();

    }

}

public class FactoryMethodPatternExample{

    public static void main(String[] args) {

        DocFactory docx = new WordDocFactory();

        Doc x = docx.createDoc();

        x.readDoc();

        DocFactory pdf = new PDFDocFactory();

        Doc y = pdf.createDoc();

        y.readDoc();

        DocFactory xlxs = new ExcelDocFactory();

        Doc z = xlxs.createDoc();

        z.readDoc();

    }

}

output:



Exercise 3: Implementing the Builder Pattern

code:

class Computer{

    String CPU;

    int RAM;

    int Storage;

    String GPU;

    String Screen;

    private Computer (Builder b){

        this.CPU = b.CPU;

        this.RAM = b.RAM;

        this.Storage = b.Storage;

        this.GPU = b.GPU;

        this.Screen = b.Screen;

    }

    void specs(){

        System.out.println("CPU : "+CPU);

        System.out.println("RAM : "+RAM+" GB");

        System.out.println("Storage : "+Storage+" GB");

        System.out.println("GPU : "+GPU);

        System.out.println("Screen : "+Screen);

    }

    static class Builder {

        private String CPU;

        private int RAM;

        private int Storage;

        private String GPU;

        private String Screen;

        Builder setCPU(String x){

            this.CPU = x;

            return this;

        }

        Builder setRAM(int x){

            this.RAM = x;

            return this;

        }

        Builder setStorage(int x){

            this.Storage = x;

            return this;

        }

        Builder setGPU(String x){

            this.GPU = x;

            return this;

        }

        Builder setScreen(String x){

            this.Screen = x;

            return this;

        }

        public Computer build(){

            return new Computer(this);

        }

    }

}

public class BuilderPatternExample {

    public static void main(String[] args) {

        Computer x = new Computer.Builder()

        .setCPU("12600k")

        .setRAM(16)

        .setStorage(512)

        .setGPU("RTX 3060")

        .setScreen("1080p")

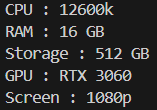
        .build();

        x.specs();

    }

}

output:



Exercise 4: Implementing the Adapter Pattern

Code:

interface PaymentProcessor {

    void processPayment(double amt);

}

class BillDeskGateway{

    void BillDeskTransfer(double amt){

        System.out.println("Transfered "+amt+" using BillDesk");

    }

}

class RazorPayGateway{

    void RazorPayTransfer(double amt){

        System.out.println("Transfered "+amt+" using RazorPay");

    }

}

class BillDeskAdapter implements PaymentProcessor{

    BillDeskGateway bdg;

    BillDeskAdapter(BillDeskGateway x){

        bdg=x;

    }

    public void processPayment(double amt){

        bdg.BillDeskTransfer(amt);

    }

}

class RazorPayAdapter implements PaymentProcessor{

    RazorPayGateway rpg;

    RazorPayAdapter(RazorPayGateway x){

        rpg=x;

    }

    public void processPayment(double amt){

        rpg.RazorPayTransfer(amt);

    }

}

public class AdapterPatternExample {

    public static void main(String[] args) {

        PaymentProcessor a=new BillDeskAdapter(new BillDeskGateway());

        a.processPayment(50000);

        PaymentProcessor b=new RazorPayAdapter(new RazorPayGateway());

        b.processPayment(75000);

    }

}

Output:

