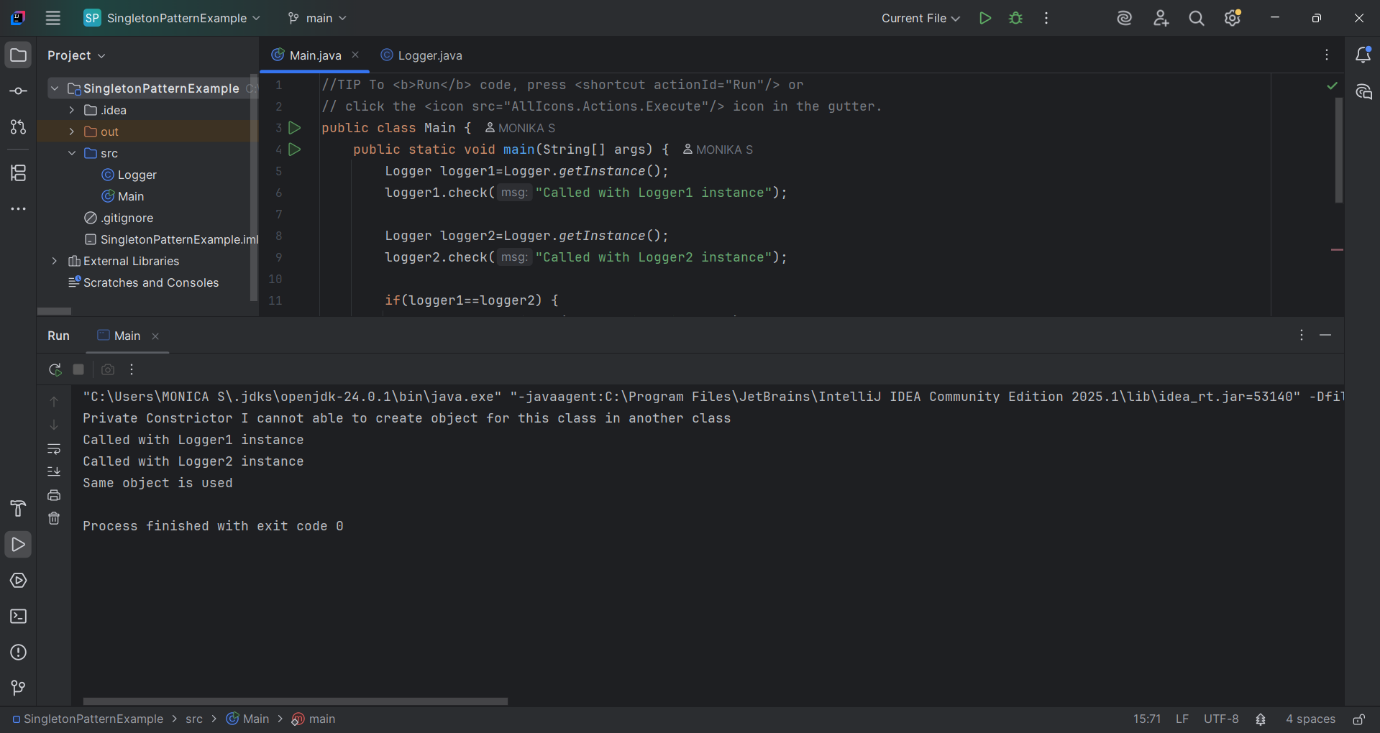
**1. SingletonPatternExample**

**Main.java**  
public class Main {  
 public static void main(String[] args) {  
 Logger logger1=Logger.*getInstance*();  
 logger1.check("Called with Logger1 instance");  
 Logger logger2=Logger.*getInstance*();  
 logger2.check("Called with Logger2 instance");  
 if(logger1==logger2) {  
 System.*out*.println("Same object is used");  
 }  
 else{  
 System.*out*.println("Not a singleton object pattern program");  
 }  
 }  
}

**Logger.java**

public class Logger {  
 private static Logger *log*;  
 private Logger() {  
 System.*out*.println("Private Constrictor I cannot able to create object for this class in another class");  
 }  
 public static Logger getInstance() {  
 if(*log*==null) {  
 *log*=new Logger();  
 return *log*;  
 }  
 return *log*;  
 }  
  
 public void check(String msg) {  
 System.*out*.println(msg);  
 }  
}



**2. FactoryMethodPatternExample**

**Main.java**  
public class Main {  
 public static void main(String[] args) {  
 DocumentFactory word=new WordDocumentFactory();  
 Document wordDoc=word.createDocument();  
 wordDoc.open();  
 wordDoc.close();  
  
 DocumentFactory pdf =new PdfDocumentfactory();  
 Document pdfDoc=pdf.createDocument();  
 pdfDoc.open();  
 pdfDoc.close();

DocumentFactory excel=new ExcelDocumentFactory();  
 Document excelDoc=excel.createDocument();  
 excelDoc.open();  
 excelDoc.close();  
 }  
}

**DocumentFactory.java**

public abstract class DocumentFactory {  
 public abstract Document createDocument();  
}

**Dacument.java**

public interface Document {  
 void open();  
 void close();  
}

**ExcelDocumentFactory.java**

public class ExcelDocumentFactory extends DocumentFactory{  
 @Override  
 public Document createDocument() {  
 return new ExcelDocument();  
 }  
}

**ExcelDocument.java**

public class ExcelDocument implements Document{  
 @Override  
 public void open() {  
 System.*out*.println("Opening Excel Document");  
 }  
 @Override  
 public void close() {  
 System.*out*.println("Closing Excel Document");  
 }  
}

**PdfDocumentFactory.java**

public class PdfDocumentfactory extends DocumentFactory{  
 @Override  
 public Document createDocument() {  
 return new PdfDocument();  
 }  
}

**PdfDocument.java**

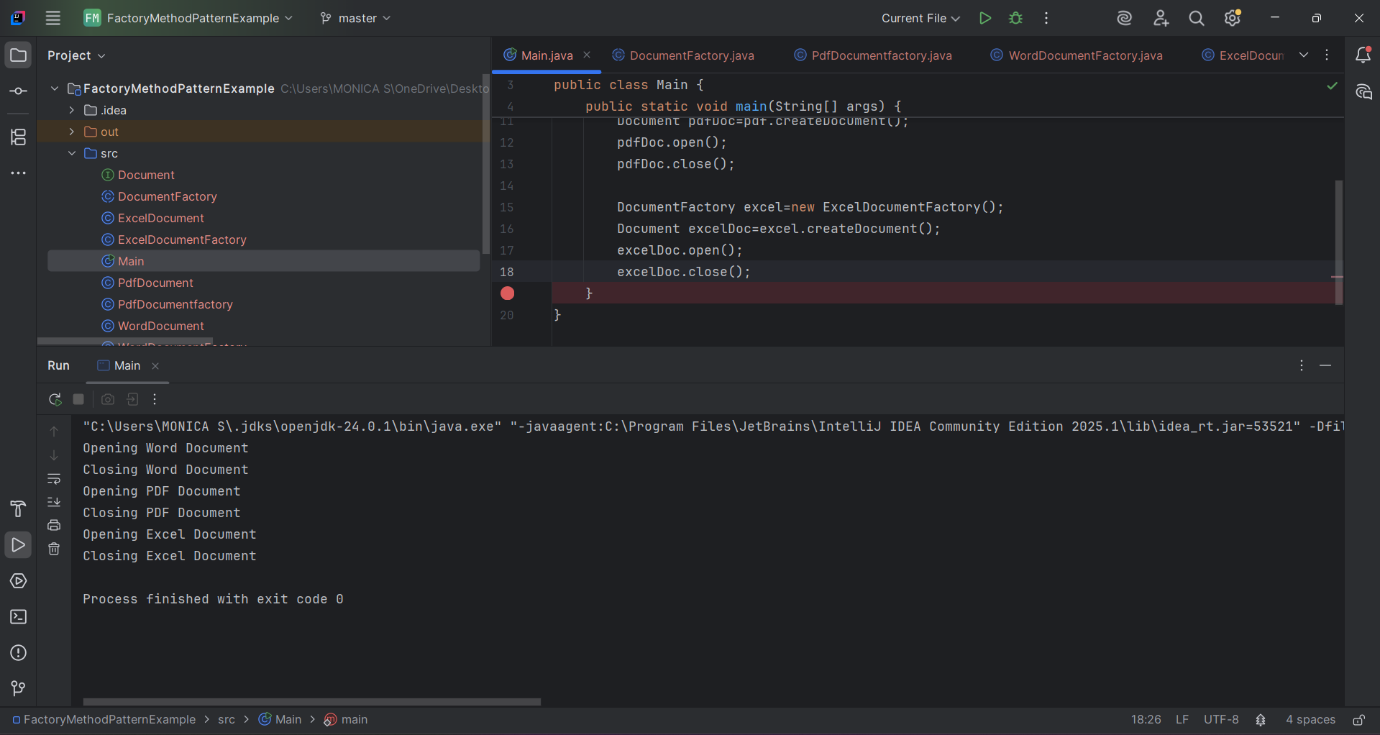
public class PdfDocument implements Document{  
 @Override  
 public void open() {   
 System.*out*.println("Opening PDF Document");  
 }  
 @Override  
 public void close() {  
 System.*out*.println("Closing PDF Document");  
 }  
}

**WordDocumentFactory.java**

public class WordDocumentFactory extends DocumentFactory{  
 @Override  
 public Document createDocument() {  
 return new WordDocument();  
 }  
}

**WordDocument.java**

public class WordDocument implements Document{  
 @Override  
 public void open() {   
 System.*out*.println("Opening Word Document");  
 }  
  
 @Override  
 public void close() {  
 System.*out*.println("Closing Word Document");  
 }  
}



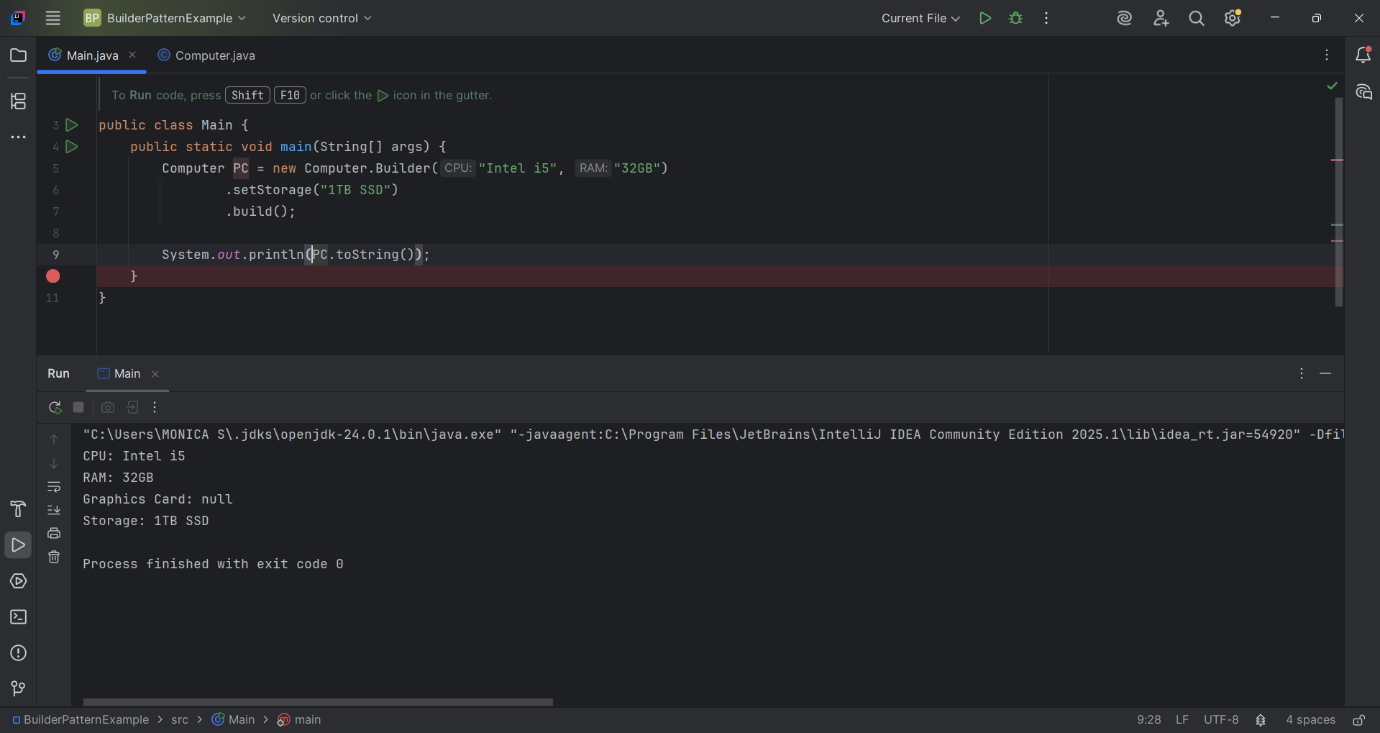
**3. BuilderPatternExample**

**Main.java**

public class Main {  
 public static void main(String[] args) {  
 Computer PC = new Computer.Builder("Intel i5", "32GB")  
 .setStorage("1TB SSD")  
 .build();  
 System.out.println(PC.toString());  
 }  
}

**Computer.java**

import javax.swing.plaf.basic.BasicButtonUI;  
  
public class Computer {  
 private String CPU;  
 private String RAM;  
 private String storage;  
 private String graphicsCard;  
  
 private Computer(Builder builder) {  
 this.CPU=builder.CPU;  
 this.RAM=builder.RAM;  
 this.graphicsCard=builder.graphicsCard;  
 this.storage=builder.storage;  
 }  
  
 public String toString() {  
 return "CPU: "+CPU+"\nRAM: "+RAM+"\nGraphics Card: "+graphicsCard+"\nStorage: "+storage;  
 }  
  
 public static class Builder{  
 private String CPU;  
 private String RAM;  
 private String storage;  
 private String graphicsCard;  
  
 public Builder(String CPU, String RAM) {  
 this.CPU=CPU;  
 this.RAM=RAM;  
 }  
  
 public Builder setStorage(String storage) {  
 this.storage=storage;  
 return this;  
 }  
  
 public Builder setGraphicsCard(String graphicsCard) {  
 this.graphicsCard=graphicsCard;  
 return this;  
 }  
  
 public Computer build() {  
 return new Computer(this);  
 }  
 }  
}

****

**4. AdapterPatternExample**

**Main.java**public class Main {  
 public static void main(String[] args) {  
 Gpay gpay=new Gpay();  
 gpay.makePayment(20000);  
 PaymentProcessor pay=new GpayAdapter(gpay);  
  
 PayPal paypal=new PayPal();  
 paypal.sendPayment(568000.31);  
 PaymentProcessor pay1=new PayPalAdapter(paypal);  
  
 }  
}

**PaymentProcessor.java**

public interface PaymentProcessor {  
 void processorPayment(double amt);  
}

**Gpay.java**

public class Gpay {  
 public void makePayment(double amount)  
 {  
 System.*out*.println("Gpay processed: "+amount);  
 }  
}

**GpayAdapter.java**

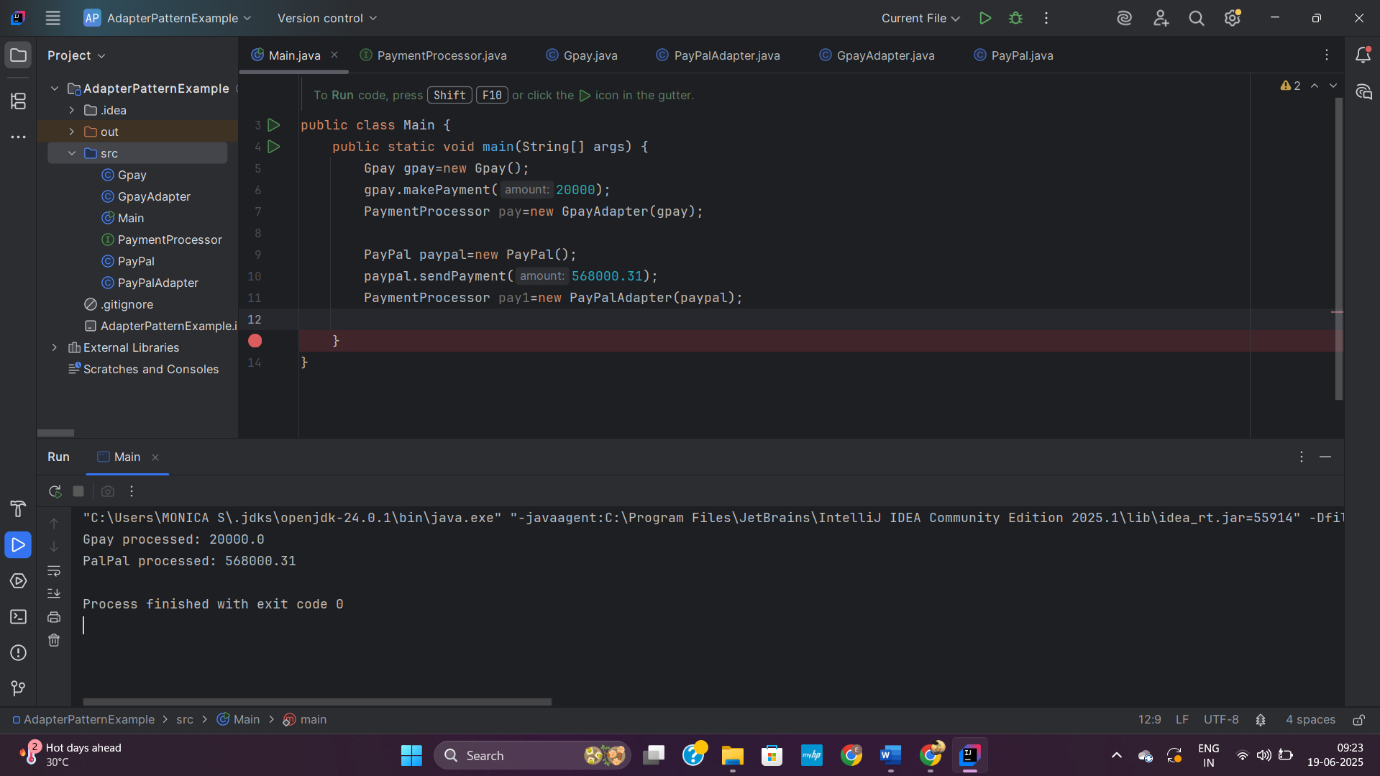
public class GpayAdapter implements PaymentProcessor {  
 Gpay gpay;  
 GpayAdapter(Gpay gpay) {  
 this.gpay=gpay;  
 }  
 @Override  
 public void processorPayment(double amt) {  
 gpay.makePayment(amt);  
 }  
}

**PayPal.java**

public class PayPal {  
 public void sendPayment(double amount) {  
 System.*out*.println("PalPal processed: "+amount);  
 }  
}

**PayPalAdapter.java**

public class PayPalAdapter implements PaymentProcessor{  
 PayPal paypal;  
 public PayPalAdapter(PayPal payPal) {  
 this.paypal=payPal;  
 }  
 @Override  
 public void processorPayment(double amt) {  
 paypal.sendPayment(amt);  
 }  
}



**5. DecoratorPatternExample**

**Main.java**  
public class Main {  
 public static void main(String[] args) {  
 Notifier notifier = new EmailNotifier();  
 notifier = new SMSNotifierDecorator(notifier);  
 notifier = new SlackNotifierDecorator(notifier);  
 notifier.send("Server is down! Immediate action required.");  
 }  
}

**Notifier.java**

public interface Notifier {  
 void send(String message);  
}

**EmailNotifier.java**

public class EmailNotifier implements Notifier{  
 @Override  
 public void send(String message) {  
 System.out.println("Email sent: "+message);  
 }  
}

**NotifierDecorator.java**

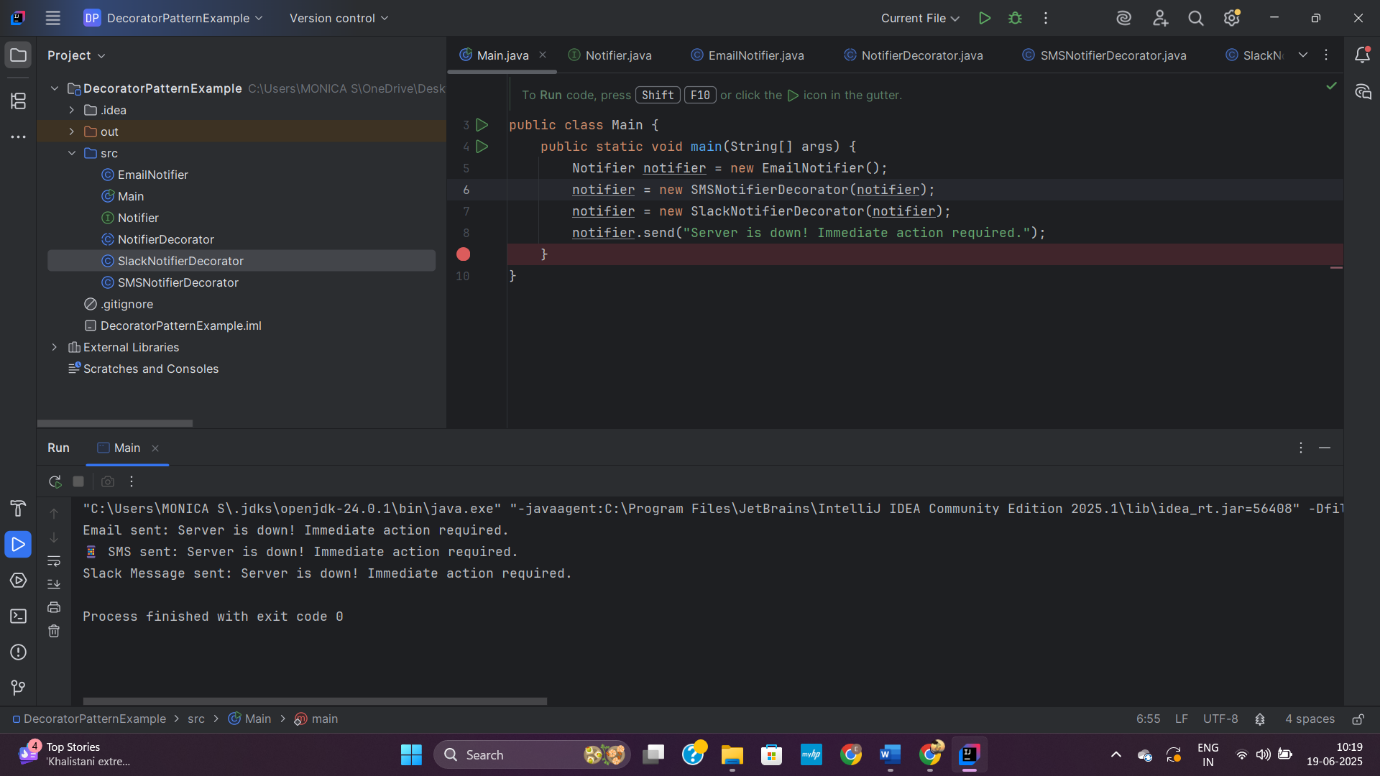
public abstract class NotifierDecorator implements Notifier{  
 protected Notifier notifiy;  
 public NotifierDecorator(Notifier notifier) {  
 this.notifiy=notifier;  
 }  
 @Override  
 public void send(String message) {  
 notifiy.send(message);  
 }  
}

**SMSNotifierDecorator.java**

public class SMSNotifierDecorator extends NotifierDecorator{  
 public SMSNotifierDecorator(Notifier notifier) {  
 super(notifier);  
 }  
 @Override  
 public void send(String message) {  
 super.send(message);   
 System.out.println("📱 SMS sent: " + message);  
 }  
}

**SlactNotifierDecorator.java**

public class SlackNotifierDecorator extends NotifierDecorator{  
 protected Notifier notify;  
 public SlackNotifierDecorator(Notifier notifier) {  
 super(notifier);  
 }  
 public void send(String message) {  
 super.send(message);  
 System.out.println("Slack Message sent: "+message);  
 }  
}



**6. ProxyPatterExample**

**Main.java**  
public class Main {  
 public static void main(String[] args) {  
 Image img1=new ProxyImage("photo1.jpg");  
 Image img2=new ProxyImage("photo2.jpg");  
  
 img1.display();  
 img1.display();  
 img2.display();  
 }  
}

**Image.java**

public interface Image {  
 public void display();  
}

**ProxyImage.java**

public class ProxyImage implements Image{  
 private String filename;  
 private RealImage real;  
  
 public ProxyImage(String filename) {  
 this.filename=filename;  
 }  
 @Override  
 public void display() {  
 if(real==null) {  
 real=new RealImage(filename);  
 }  
 real.display();  
 }  
}

**RealImage.java**

public class RealImage implements Image  
{  
 private String filename;  
 public RealImage(String filename) {  
 this.filename=filename;  
 loadFromRemoteServer();  
 }  
 private void loadFromRemoteServer() {  
 System.out.println("Loading image from remote server: " + filename);  
 }  
 @Override  
 public void display() {   
 System.out.println("Displaying image: "+filename);  
 }  
}

