**MODULE 1 EXERCISES**

**Exercise 1: Implementing the Singleton Pattern**

class Logger {

private static Logger instance;

private Logger() {

System.out.println("Logger is here");

}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void log(String message) {

System.out.println("LOG : " + message);

}

}

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("My first name is Saurav");

Logger logger2 = Logger.getInstance();

logger2.log("My last name is Jena");

if (logger1 == logger2) {

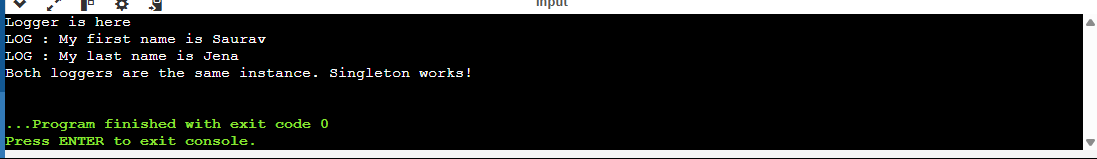
System.out.println("Both loggers are the same instance. Singleton works!");

} else {

System.out.println("Different instances! Singleton failed.");

}

}

}****

**Exercise 2: Implementing the Factory Method Pattern**

**abstract class Document {**

**public abstract void open();**

**}**

**class WordDocument extends Document {**

**@Override**

**public void open() {**

**System.out.println("Opening a Word Document.");**

**}**

**}**

**class PdfDocument extends Document {**

**@Override**

**public void open() {**

**System.out.println("Opening a PDF Document.");**

**}**

**}**

**class ExcelDocument extends Document {**

**@Override**

**public void open() {**

**System.out.println("Opening an Excel Document.");**

**}**

**}**

**// Abstract Factory**

**abstract class DocumentFactory {**

**public abstract Document createDocument();**

**}**

**// Concrete Factories**

**class WordDocumentFactory extends DocumentFactory {**

**@Override**

**public Document createDocument() {**

**return new WordDocument();**

**}**

**}**

**class PdfDocumentFactory extends DocumentFactory {**

**@Override**

**public Document createDocument() {**

**return new PdfDocument();**

**}**

**}**

**class ExcelDocumentFactory extends DocumentFactory {**

**@Override**

**public Document createDocument() {**

**return new ExcelDocument();**

**}**

**}**

**public class Main {**

**public static void main(String[] args) {**

**DocumentFactory wordFactory = new WordDocumentFactory();**

**Document word = wordFactory.createDocument();**

**word.open();**

**DocumentFactory pdfFactory = new PdfDocumentFactory();**

**Document pdf = pdfFactory.createDocument();**

**pdf.open();**

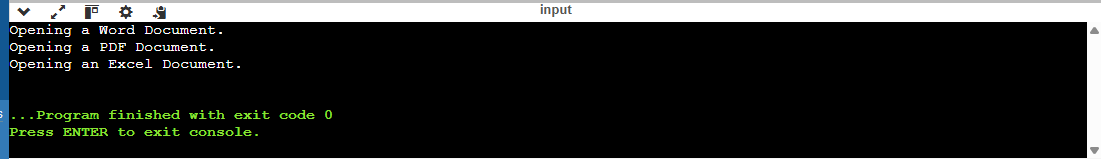
**DocumentFactory excelFactory = new ExcelDocumentFactory();**

**Document excel = excelFactory.createDocument();**

**excel.open();**

**}**

**}**

****

**3.BUILDER PATTERN EXAMPLE**

**package BuilderPatternExample;**

**public class computer {**

**private String cpu;**

**private String ram;**

**private String storage;**

**public static class builder{**

**private String cpu;**

**private String ram;**

**private String storage;**

**public builder setcpu(String cpu) {**

**this.cpu=cpu;**

**return this;**

**}**

**public builder setram(String ram ) {**

**this.ram=ram;**

**return this;**

**}**

**public builder setstorage(String storage) {**

**this.storage=storage;**

**return this;**

**}**

**public computer build() {**

**return new computer(this);**

**}**

**}**

**private computer (builder builder) {**

**this.cpu=builder.cpu;**

**this.ram=builder.ram;**

**this.storage=builder.storage;**

**}**

**public String getcpu() {**

**return cpu;**

**}**

**public String getram() {**

**return ram;**

**}**

**public String getstorage() {**

**return storage;**

**}**

**public void testshow() {**

**System.*out*.println("cpu"+cpu);**

**System.*out*.println("ram"+ram);**

**System.*out*.println("storage"+storage);**

**}**

**}**

**package BuilderPatternExample;**

**public class main {**

**public static void main(String[] args) {**

**computer basicComputer = new computer.builder()**

**.setcpu("Intel i3")**

**.setram("4GB")**

**.setstorage("256GB SSD")**

**.build();**

**computer gamingComputer = new computer.builder()**

**.setcpu("Intel i9")**

**.setram("32GB")**

**.setstorage("1TB SSD")**

**.build();**

**System.*out*.println("=== Basic Computer ===");**

**System.*out*.println("CPU: " + basicComputer.getcpu());**

**System.*out*.println("RAM: " + basicComputer.getram());**

**System.*out*.println("Storage: " + basicComputer.getstorage());**

**System.*out*.println("\n=== Gaming Computer ===");**

**System.*out*.println("CPU: " + gamingComputer.getcpu());**

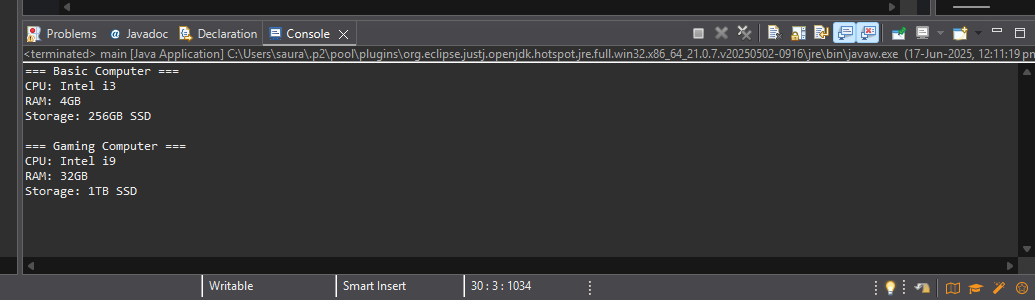
**System.*out*.println("RAM: " + gamingComputer.getram());**

**System.*out*.println("Storage: " + gamingComputer.getstorage());**

**}**

**}**

**OUTPUT**

****

**4.ADAPTER PATTERN**

**package AdapterPatternExample;**

**public interface paymentprocessor {**

**void processpayment(double amount);**

**}**

**public class paytmgateway {**

**public void makepayment(double amount) {**

**System.*out*.println("payment of $ "+ amount+"made via paytm.");**

**}**

**}**

**public void makepayment(double amount) {**

**System.*out*.println("payment of $ "+ amount+"processed using razorpay .");**

**}**

**}**

**public class paytmAdapter implements paymentprocessor {**

**private paytmgateway paytm;**

**public paytmAdapter (paytmgateway paytm) {**

**this.paytm=paytm;**

**}**

**public void processpayment(double amount) {**

**paytm.makepayment(amount);**

**}**

**}**

**public class razorAdapter implements paymentprocessor {**

**private razorpaygateway razor;**

**public razorAdapter(razorpaygateway razor) {**

**this.razor=razor;**

**}**

**public void processpayment(double amount) {**

**razor.makepayment(amount);**

**}**

**}**

**public class Main {**

**public static void main(String[] args) {**

**// Using Stripe through Adapter**

**razorpaygateway razor = new razorpaygateway();**

**paymentprocessor razorAdapter = new razorAdapter(razor);**

**razorAdapter.processpayment(150.0);**

**// Using PayPal through Adapter**

**paytmgateway paytm = new paytmgateway();**

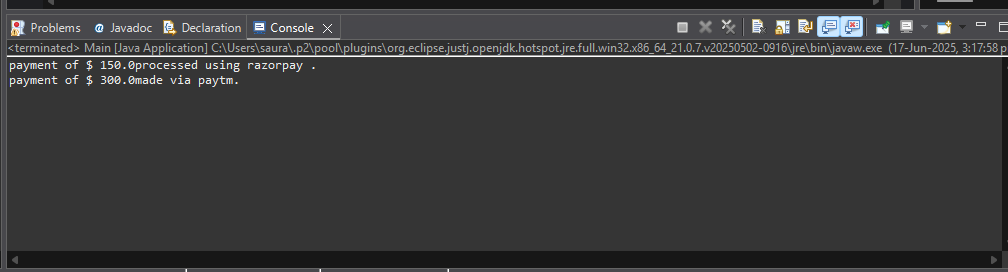
**paymentprocessor paytmAdapter = new paytmAdapter(paytm);**

**paytmAdapter.processpayment(300.0);**

**}**

**}**

**OUTPUT**

****

**Exercise 5: Implementing the Decorator Pattern**

**package DecoratorPattternExample;**

**public class emailnotifier implements notifier {**

**public void send (String message) {**

**System.*out*.println("sending email : "+ message);**

**}**

**}**

**package DecoratorPattternExample;**

**public interface notifier {**

**void send(String message);**

**}**

**package DecoratorPattternExample;**

**public abstract class notifierdecorater implements notifier {**

**protected notifier wrappednotifier;**

**public notifierdecorater(notifier notifier) {**

**this.wrappednotifier=notifier;**

**}**

**public void send (String message) {**

**wrappednotifier.send(message);**

**}**

**}**

**package DecoratorPattternExample;**

**public class slacknotifierdecorater extends notifierdecorater {**

**public slacknotifierdecorater(notifier notifier) {**

**super(notifier);**

**}**

**public void send(String message) {**

**super.send(message);**

**sendslack(message);**

**}**

**private void sendslack(String message) {**

**System.*out*.println("sending slack messsage "+message);**

**}**

**}**

**package DecoratorPattternExample;**

**public class smsnotifierdecorater extends notifierdecorater {**

**public smsnotifierdecorater(notifier notifier) {**

**super(notifier);**

**}**

**public void send(String message) {**

**super.send(message);**

**sendsms(message);**

**}**

**private void sendsms(String message) {**

**System.*out*.println("sending sms "+message);**

**}**

**}**

**package DecoratorPattternExample;**

**public class smsnotifierdecorater extends notifierdecorater {**

**public smsnotifierdecorater(notifier notifier) {**

**super(notifier);**

**}**

**public void send(String message) {**

**super.send(message);**

**sendsms(message);**

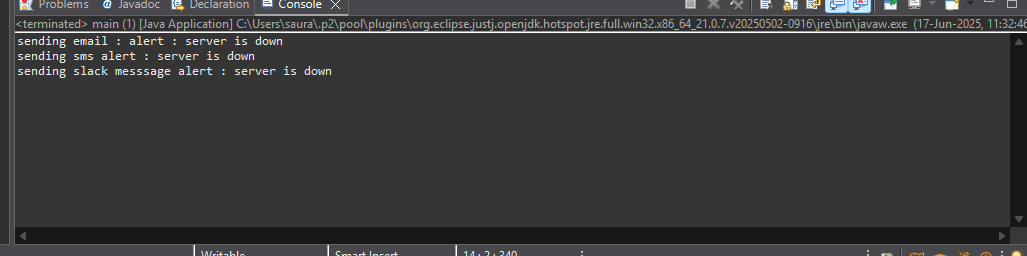
**}**

**private void sendsms(String message) {**

**System.*out*.println("sending sms "+message);**

**}**

**}**

****

**Exercise 6: Implementing the Proxy Pattern**

**package ProxyPatternExample;**

**public interface image {**

**void display();**

**}**

**package ProxyPatternExample;**

**public class proxyimage implements image{**

**private String filename;**

**private realimage realimage;**

**public proxyimage(String filename) {**

**this.filename=filename;**

**}**

**public void display() {**

**if(realimage == null) {**

**realimage=new realimage(filename);**

**}**

**realimage.display();**

**}**

**}**

**package ProxyPatternExample;**

**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 "+filename);**

**}**

**public void display() {**

**System.*out*.println("displaying image : "+filename);**

**}**

**}**

**package ProxyPatternExample;**

**public class main {**

**public static void main(String[] args) {**

**// TODO Auto-generated method stub**

**image image1=new proxyimage("photo1.jpg");**

**image image2=new proxyimage("photo2.jpg");**

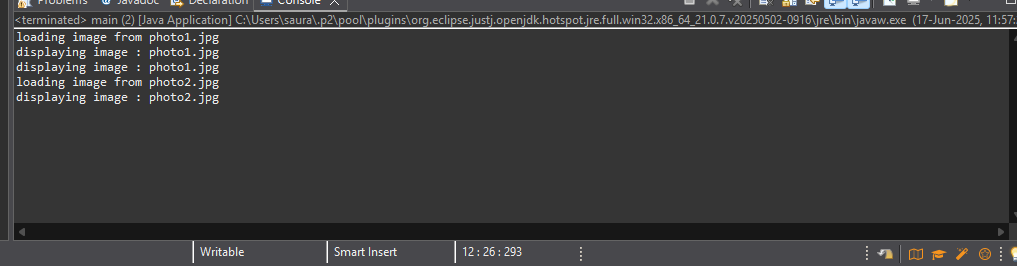
**image1.display();**

**image1.display();**

**image2.display();**

**}**

**}**

****

**Exercise 7: Implementing the Observer Pattern**

**package ObserverPatternExample;**

**public interface Stock {**

**void registerObserver(Observer o);**

**void removeObserver(Observer o);**

**void notifyObservers();**

**}**

**package ObserverPatternExample;**

**public interface Observer {**

**void update(String stockName, double price);**

**}**

**package ObserverPatternExample;**

**import java.util.ArrayList;**

**import java.util.List;**

**public class StockMarket implements Stock {**

**private List<Observer> observers = new ArrayList<>();**

**private String stockName;**

**private double stockPrice;**

***@Override***

**public void registerObserver(Observer o) {**

**observers.add(o);**

**}**

***@Override***

**public void removeObserver(Observer o) {**

**observers.remove(o);**

**}**

***@Override***

**public void notifyObservers() {**

**for (Observer o : observers) {**

**o.update(stockName, stockPrice);**

**}**

**}**

**public void setStock(String stockName, double stockPrice) {**

**this.stockName = stockName;**

**this.stockPrice = stockPrice;**

**notifyObservers();**

**}**

**}**

**package ObserverPatternExample;**

**public class WebApp implements Observer {**

**private String name;**

**public WebApp(String name) {**

**this.name = name;**

**}**

***@Override***

**public void update(String stockName, double price) {**

**System.*out*.println("WebApp [" + name + "] - Stock Update: " + stockName + " is now ₹" + price);**

**}**

**}**

**package ObserverPatternExample;**

**public class MobileApp implements Observer {**

**private String name;**

**public MobileApp(String name) {**

**this.name = name;**

**}**

***@Override***

**public void update(String stockName, double price) {**

**System.*out*.println("MobileApp [" + name + "] - Stock Update: " + stockName + " is now ₹" + price);**

**}**

**}**

**package ObserverPatternExample;**

**public class Main {**

**public static void main(String[] args) {**

**StockMarket stockMarket = new StockMarket();**

**Observer mobile1 = new MobileApp("MobileUser1");**

**Observer web1 = new WebApp("WebUserA");**

**stockMarket.registerObserver(mobile1);**

**stockMarket.registerObserver(web1);**

**stockMarket.setStock("TCS", 3790.45);**

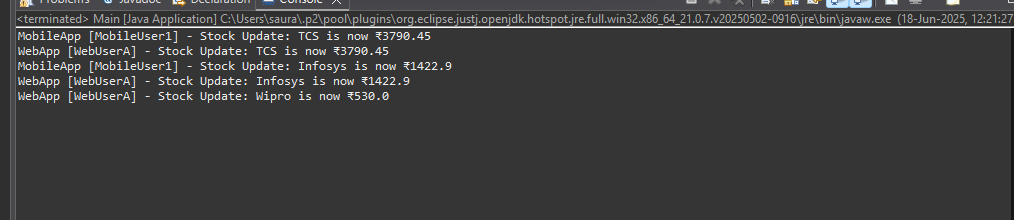
**stockMarket.setStock("Infosys", 1422.90);**

**stockMarket.removeObserver(mobile1);**

**stockMarket.setStock("Wipro", 530.00);**

**}**

**}**

****

**Exercise 8: Implementing the Strategy Pattern**

**package StrategyPatternExample;**

**public interface PaymentStrategy {**

**void pay(double amount);**

**}**

**package StrategyPatternExample;**

**public class PaymentContext {**

**private PaymentStrategy paymentStrategy;**

**public void setPaymentStrategy(PaymentStrategy paymentStrategy) {**

**this.paymentStrategy = paymentStrategy;**

**}**

**public void payAmount(double amount) {**

**if (paymentStrategy == null) {**

**System.*out*.println("Payment method not selected.");**

**} else {**

**paymentStrategy.pay(amount);**

**}**

**}**

**}**

**package StrategyPatternExample;**

**public class CreditCardPayment implements PaymentStrategy {**

**private String cardNumber;**

**private String name;**

**public CreditCardPayment(String cardNumber, String name) {**

**this.cardNumber = cardNumber;**

**this.name = name;**

**}**

***@Override***

**public void pay(double amount) {**

**System.*out*.println("Paid ₹" + amount + " using Credit Card [" + cardNumber + "] - " + name);**

**}**

**}**

**package StrategyPatternExample;**

**public class PayPalPayment implements PaymentStrategy {**

**private String email;**

**public PayPalPayment(String email) {**

**this.email = email;**

**}**

***@Override***

**public void pay(double amount) {**

**System.*out*.println("Paid ₹" + amount + " using PayPal [" + email + "]");**

**}**

**}**

**package StrategyPatternExample;**

**public class Main {**

**public static void main(String[] args) {**

**PaymentContext context = new PaymentContext();**

**context.setPaymentStrategy(new CreditCardPayment("1234-5678-9876", "Saurav Jena"));**

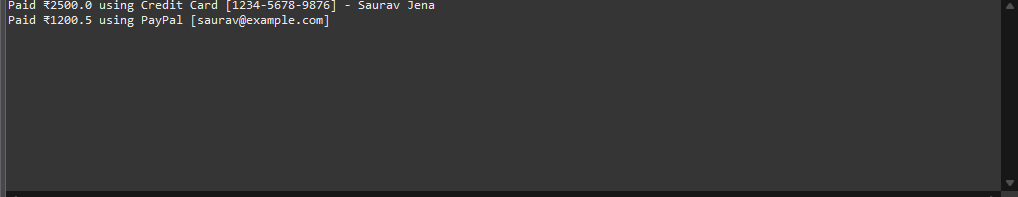
**context.payAmount(2500.0);**

**context.setPaymentStrategy(new PayPalPayment("saurav@example.com"));**

**context.payAmount(1200.5);**

**}**

**}**

****

**Exercise 10: Implementing the MVC Pattern**

**package MVCPatternExample;**

**public class Student {**

**private String name;**

**private String id;**

**private String grade;**

**public Student(String name, String id, String grade) {**

**this.name = name;**

**this.id = id;**

**this.grade = grade;**

**}**

**public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**public String getId() {**

**return id;**

**}**

**public void setId(String id) {**

**this.id = id;**

**}**

**public String getGrade() {**

**return grade;**

**}**

**public void setGrade(String grade) {**

**this.grade = grade;**

**}**

**}**

**package MVCPatternExample;**

**public class StudentView {**

**public void displayStudentDetails(String name, String id, String grade) {**

**System.*out*.println("---- Student Details ----");**

**System.*out*.println("Name : " + name);**

**System.*out*.println("ID : " + id);**

**System.*out*.println("Grade: " + grade);**

**System.*out*.println("-------------------------");**

**}**

**}**

**package MVCPatternExample;**

**public class StudentController {**

**private Student model;**

**private StudentView view;**

**public StudentController(Student model, StudentView view) {**

**this.model = model;**

**this.view = view;**

**}**

**public void setStudentName(String name) {**

**model.setName(name);**

**}**

**public String getStudentName() {**

**return model.getName();**

**}**

**public void setStudentId(String id) {**

**model.setId(id);**

**}**

**public String getStudentId() {**

**return model.getId();**

**}**

**public void setStudentGrade(String grade) {**

**model.setGrade(grade);**

**}**

**public String getStudentGrade() {**

**return model.getGrade();**

**}**

**public void updateView() {**

**view.displayStudentDetails(model.getName(), model.getId(), model.getGrade());**

**}**

**}**

**package MVCPatternExample;**

**public class Main {**

**public static void main(String[] args) {**

**Student student = new Student("Alice", "SOA123", "A");**

**StudentView view = new StudentView();**

**StudentController controller = new StudentController(student, view);**

**controller.updateView();**

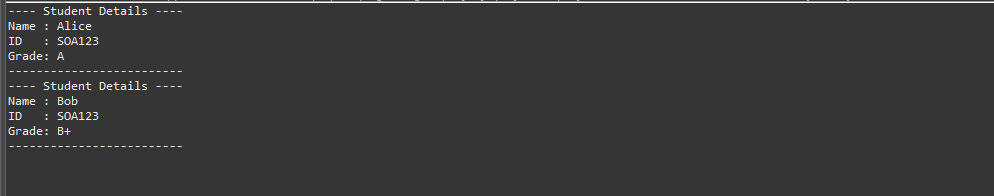
**controller.setStudentName("Bob");**

**controller.setStudentGrade("B+");**

**controller.updateView();**

**}**

**}**

****

**Exercise 11: Implementing Dependency Injection**

**package DependencyInjectionExample;**

**public interface CustomerRepository {**

**Customer findCustomerById(String id);**

**}**

**package DependencyInjectionExample;**

**public class Customer {**

**private String id;**

**private String name;**

**public Customer(String id, String name) {**

**this.id = id;**

**this.name = name;**

**}**

**public String getId() {**

**return id;**

**}**

**public String getName() {**

**return name;**

**}**

**}**

**package DependencyInjectionExample;**

**public class CustomerRepositoryImpl implements CustomerRepository {**

***@Override***

**public Customer findCustomerById(String id) {**

**// Simulating fetching from database**

**return new Customer(id, "Saurav Kumar Jena");**

**}**

**}**

**package DependencyInjectionExample;**

**public class CustomerService {**

**private CustomerRepository customerRepository;**

**// Constructor Injection**

**public CustomerService(CustomerRepository customerRepository) {**

**this.customerRepository = customerRepository;**

**}**

**public void displayCustomer(String id) {**

**Customer customer = customerRepository.findCustomerById(id);**

**System.*out*.println("Customer ID: " + customer.getId());**

**System.*out*.println("Customer Name: " + customer.getName());**

**}**

**}**

**package DependencyInjectionExample;**

**public class Main {**

**public static void main(String[] args) {**

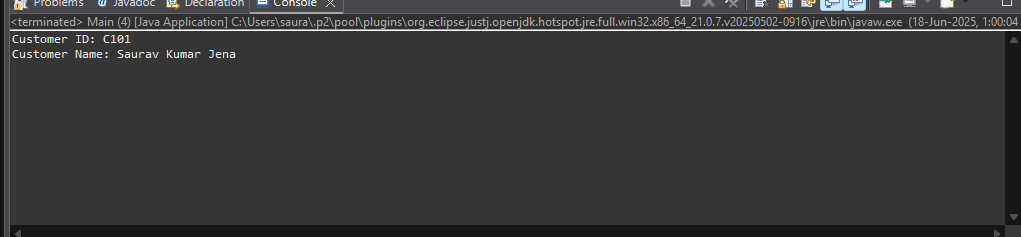
**CustomerRepository repository = new CustomerRepositoryImpl();**

**CustomerService service = new CustomerService(repository);**

**service.displayCustomer("C101");**

**}**

**}**

****