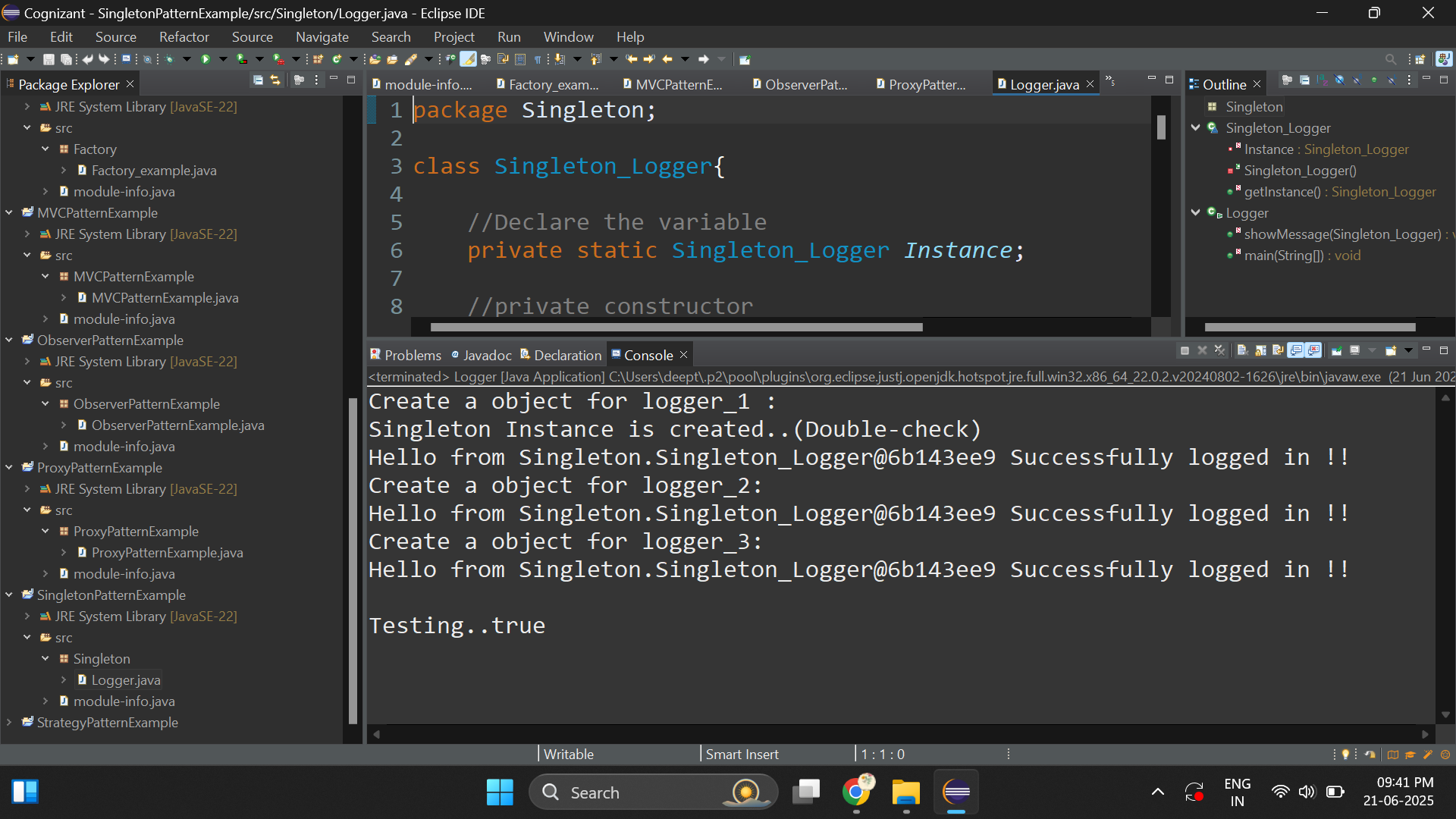
**Exercise 1: Implementing the Singleton Pattern**

class Singleton\_Logger{  
  
 //Declare the variable  
 private static Singleton\_Logger Instance;  
  
 //private constructor  
 private Singleton\_Logger() {  
 System.out.println("Singleton Instance is created..(Double-check)");  
 }  
 //getinstance - (Thread safe method)  
 public static Singleton\_Logger getInstance() {  
 if(Instance==null) {  
 synchronized(Singleton\_Logger.class){  
 if(Instance==null) {  
 Instance=new Singleton\_Logger();  
 }  
 }  
 }  
 return Instance;  
 }  
  
}  
  
  
  
public class Logger {  
 public static void showMessage(Singleton\_Logger l) {  
 System.out.println("Hello from " + l + " Successfully logged in !!");  
  
  
 }  
  
 public static void main(String[] args) {  
 System.out.println("Create a object for logger\_1 :");  
 Singleton\_Logger person\_1=Singleton\_Logger.getInstance();  
 showMessage(person\_1) ;  
  
 System.out.println("Create a object for logger\_2:");  
 Singleton\_Logger person\_2=Singleton\_Logger.getInstance();  
 showMessage(person\_2) ;  
  
  
 System.out.println("Create a object for logger\_3:");  
 Singleton\_Logger person\_3=Singleton\_Logger.getInstance();  
 showMessage(person\_3) ;  
  
  
 System.out.println();  
 System.out.println("Testing.."+ (person\_1==person\_2));  
  
  
  
  
 }  
  
  
}

**Output:**



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

package Factory;

//super class

interface Select\_Document\_Type{

String prepare();

String completed();

}

//subclasses

class Word implements Select\_Document\_Type{

public String prepare() {

return "Preparing document (Word format)!";

}

public String completed() {

return "Successfully completed.";

}

}

class pdf implements Select\_Document\_Type{

public String prepare() {

return "Preparing document (Pdf format)!";

}

public String completed() {

return "Successfully completed.";

}

}

class excel implements Select\_Document\_Type{

public String prepare() {

return "Preparing document (excel format)!";

}

public String completed() {

return "Successfully completed.";

}

}

//DocumentFactory -> Object creation (don't need for modification)

abstract class DocumentFactory{

abstract Select\_Document\_Type createDocument();

}

class WordFactory extends DocumentFactory{

public Select\_Document\_Type createDocument() {

return new Word(); //word object

}

}

class pdfFactory extends DocumentFactory{

public Select\_Document\_Type createDocument() {

return new pdf(); //pdf object

}

}

class excelFactory extends DocumentFactory{

public Select\_Document\_Type createDocument() {

return new excel(); //excel object

}

}

public class Factory\_example {

public static void main(String[] args) {

DocumentFactory factory=new WordFactory(); //select the factory

Select\_Document\_Type type=factory.createDocument(); //start to create the document

System.***out***.println(type.prepare()); //document is preparing

System.***out***.println(type.completed());

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

DocumentFactory factory\_2=new pdfFactory();

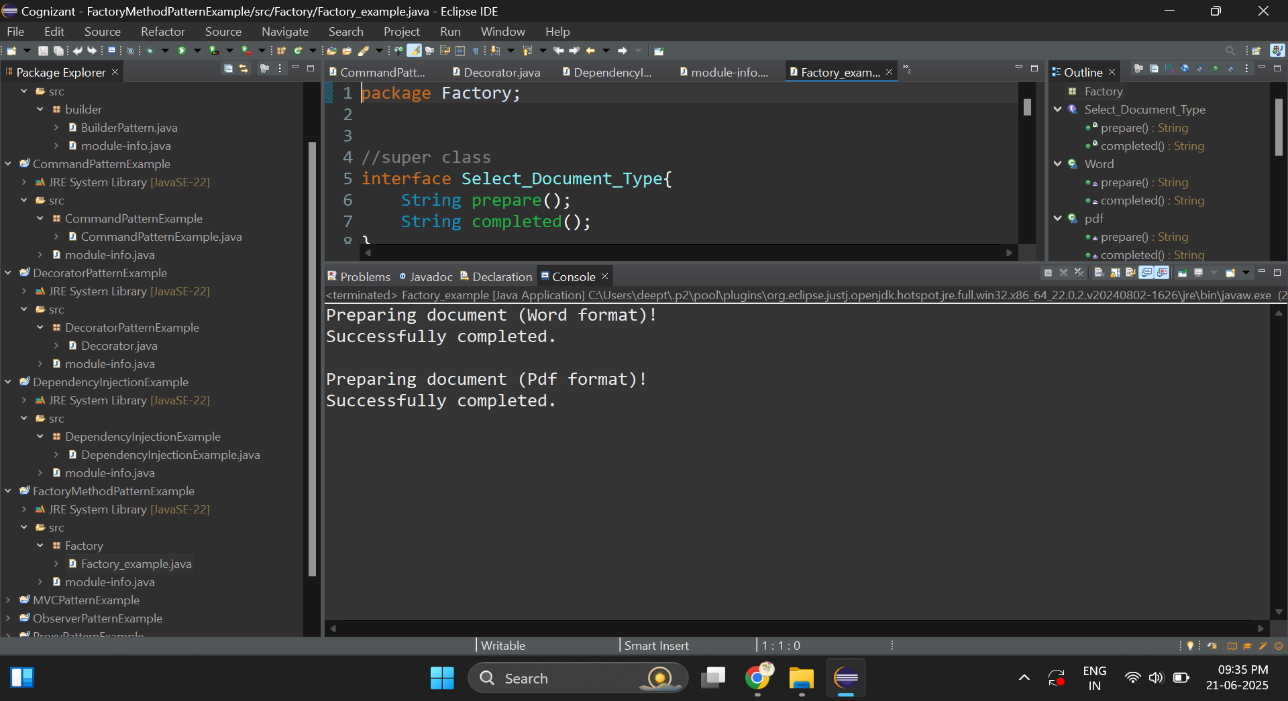
Select\_Document\_Type type\_2=factory\_2.createDocument();

System.***out***.println(type\_2.prepare());

System.***out***.println(type\_2.completed());

}

}

**Output:**

­­

**Exercise 3: Implementing the Builder Pattern**

package \_3\_BuilderPatternExample;

class Computer{

private final String HDD;

private final String CPU;

private final String RAM;

//Optional features

private final String Bluetooth\_Adapter;

private final String Finger\_Print\_Scanner;

private final String TouchScreen;

//create private constructor

private Computer(String HDD,String CPU,String RAM,

String Bluetooth\_Adapter,String Finger\_Print\_Scanner,

String TouchScreen) {

this.HDD=HDD;

this.CPU=CPU;

this.RAM=RAM;

this.Bluetooth\_Adapter=Bluetooth\_Adapter;

this.Finger\_Print\_Scanner=Finger\_Print\_Scanner;

this.TouchScreen=TouchScreen;

}

public String toString() {

return "Computer Configuration:\n" +

"HDD: " + HDD + "\n" +

"CPU: " + CPU + "\n" +

"RAM: " + RAM + "\n" +

"Bluetooth Adapter: " + (Bluetooth\_Adapter != null ? Bluetooth\_Adapter : "Not Included") + "\n" +

"Fingerprint Scanner: " + (Finger\_Print\_Scanner != null ? Finger\_Print\_Scanner : "Not Included") + "\n" +

"Touch Screen: " + (TouchScreen != null ? TouchScreen : "Not Included");

}

//create computerBuilder

public static class ComputerBuilder{

private final String HDD;

private final String CPU;

private final String RAM;

//Optional features

private String Bluetooth\_Adapter;

private String Finger\_Print\_Scanner;

private String TouchScreen;

//builder basic computer

public ComputerBuilder(String HDD,String CPU,String RAM) {

this.HDD=HDD;

this.CPU=CPU;

this.RAM=RAM;

}

//enable Bluetooth\_Adapter

public ComputerBuilder enableBluetooth\_Adapter(String Bluetooth\_Adapter) {

this.Bluetooth\_Adapter=Bluetooth\_Adapter;

return this;

}

// enable Finger\_Print\_Scanner

public ComputerBuilder enableFinger\_Print\_Scanner(String Finger\_Print\_Scanner ) {

this.Finger\_Print\_Scanner=Finger\_Print\_Scanner ;

return this;

}

//enable TouchScreen

public ComputerBuilder enableTouchScreen(String TouchScreen) {

this.TouchScreen=TouchScreen ;

return this;

}

//build

public Computer build() {

return new Computer(HDD,CPU,RAM,Bluetooth\_Adapter,Finger\_Print\_Scanner,TouchScreen);

}

}

}

public class BuilderPattern {

public static void main(String[] args) {

//create GamingComputer

Computer gamingComputer=new Computer.ComputerBuilder("1 TB","Intel i5","16 GB").

enableBluetooth\_Adapter("Enabled").

enableTouchScreen("Enabled").build();

System.***out***.println(gamingComputer);

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

//create BasicComputer

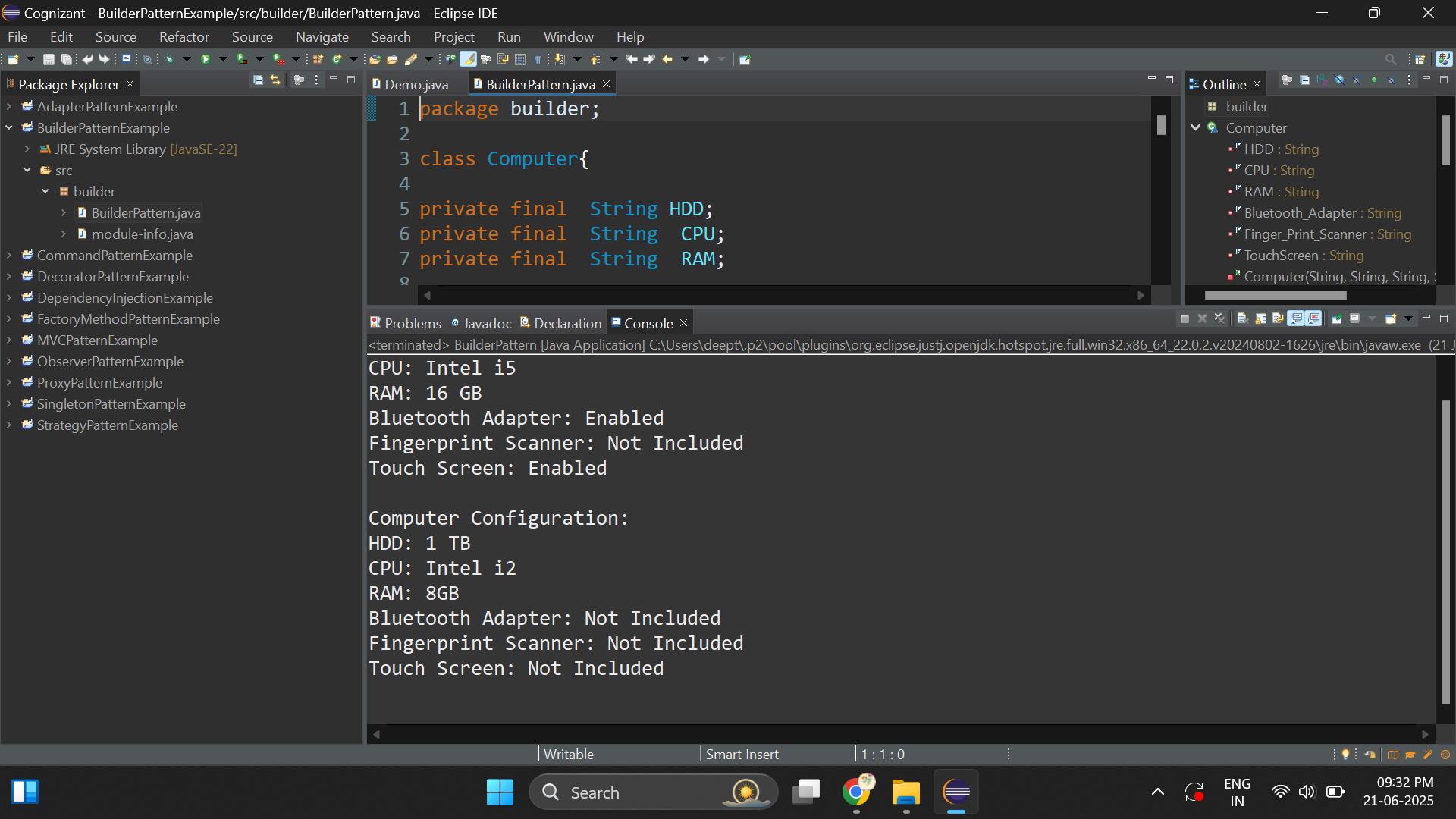
Computer basic=new Computer.ComputerBuilder("1 TB","Intel i2","8GB").build();

System.***out***.println(basic);

}

}

**Output:**



**Exercise 4: Implementing the Adapter Pattern**

package \_4\_AdapterPatternExample;

interface PaymentProcessor{

void processPayment();

}

class Gpay implements PaymentProcessor{

public String name;

public int number;

public double amount;

public Gpay(String name, int number,double amount) {

this.name=name;

this.number=number;

this.amount=amount;

}

public String getName() {

return name;

}

public int getNumber() {

return number;

}

public double getAmount() {

return amount;

}

public void processPayment() {

System.***out***.println("Payment Method - Gpay");

System.***out***.println("Payment details :");

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

System.***out***.println("Number: "+getNumber());

System.***out***.println("Amount: "+getAmount());

}

}

class Phonepe implements PaymentProcessor{

public String P\_name;

public int P\_number;

public double P\_amount;

public Phonepe(String P\_name,int P\_number,double P\_amount) {

this.P\_name=P\_name;

this.P\_amount=P\_amount;

this.P\_number=P\_number;

}

public String getP\_name() {

return P\_name;

}

public int getP\_number() {

return P\_number;

}

public double getP\_amount() {

return P\_amount;

}

public void processPayment()

{

System.***out***.println("Payment Method - Phonepe");

System.***out***.println("Payment details :");

System.***out***.println("Name: "+getP\_name());

System.***out***.println("Number: "+getP\_number());

System.***out***.println("Amount: "+getP\_amount());

}

}

class PhonepeAdapter implements PaymentProcessor {

Phonepe phonepe;

public PhonepeAdapter(Phonepe phonepe){

this.phonepe=phonepe;

}

public String getName() {

return phonepe.getP\_name();

}

public int getNumber() {

return phonepe.P\_number;

}

public double getAmount() {

return phonepe.getP\_amount();

}

public void processPayment() {

phonepe.processPayment();

}

}

public class Demo {

public static void main(String[] args) {

PaymentProcessor gpay=new Gpay("Deepthikha",43827929,2000.0);

gpay.processPayment();

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

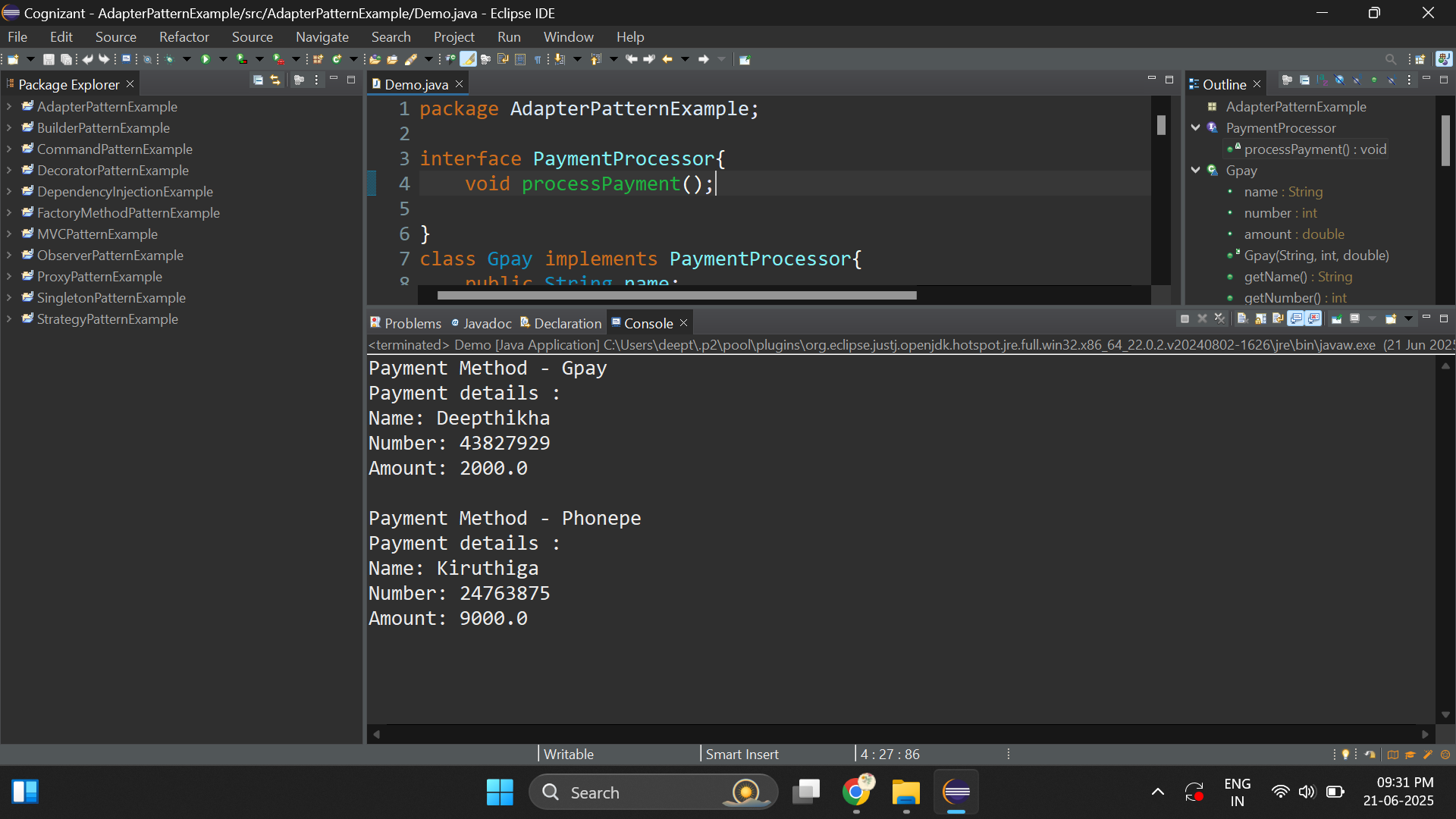
PaymentProcessor phonepe=new PhonepeAdapter(new Phonepe ("Kiruthiga",24763875,9000));

phonepe.processPayment();

}

}

**Output:**



**Exercise 5: Implementing the Decorator Pattern**

package \_5\_DecoratorPatternExample;

interface Notifier{

public void send();

}

class EmailNotifier implements Notifier{

public void send() {

System.***out***.println("Sending notification - (Email)!!");

}

}

abstract class NotifierDecorator implements Notifier{

Notifier notifier;

NotifierDecorator(Notifier notifier){

this.notifier=notifier;

}

public void send() {

notifier.send();

}

}

class SMSNotifierDecorator extends NotifierDecorator {

SMSNotifierDecorator(Notifier notifier){

super(notifier);

}

public void send() {

super.send();

System.***out***.println("Adding SMSNotifierDecorator");

}

}

class SlackNotifierDecorator extends NotifierDecorator {

SlackNotifierDecorator (Notifier notifier){

super(notifier);

}

public void send() {

super.send();

System.***out***.println("Adding SlackNotifierDecorator");

}

}

public class Decorator {

public static void main(String[] args) {

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

Notifier notify=new EmailNotifier();

Notifier notify2=new EmailNotifier();

//add SlackNotifierDecorator

notify=new SlackNotifierDecorator(notify);

notify.send();

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

//add SMSNotifierDecorator

notify=new SMSNotifierDecorator(notify);

notify.send();

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

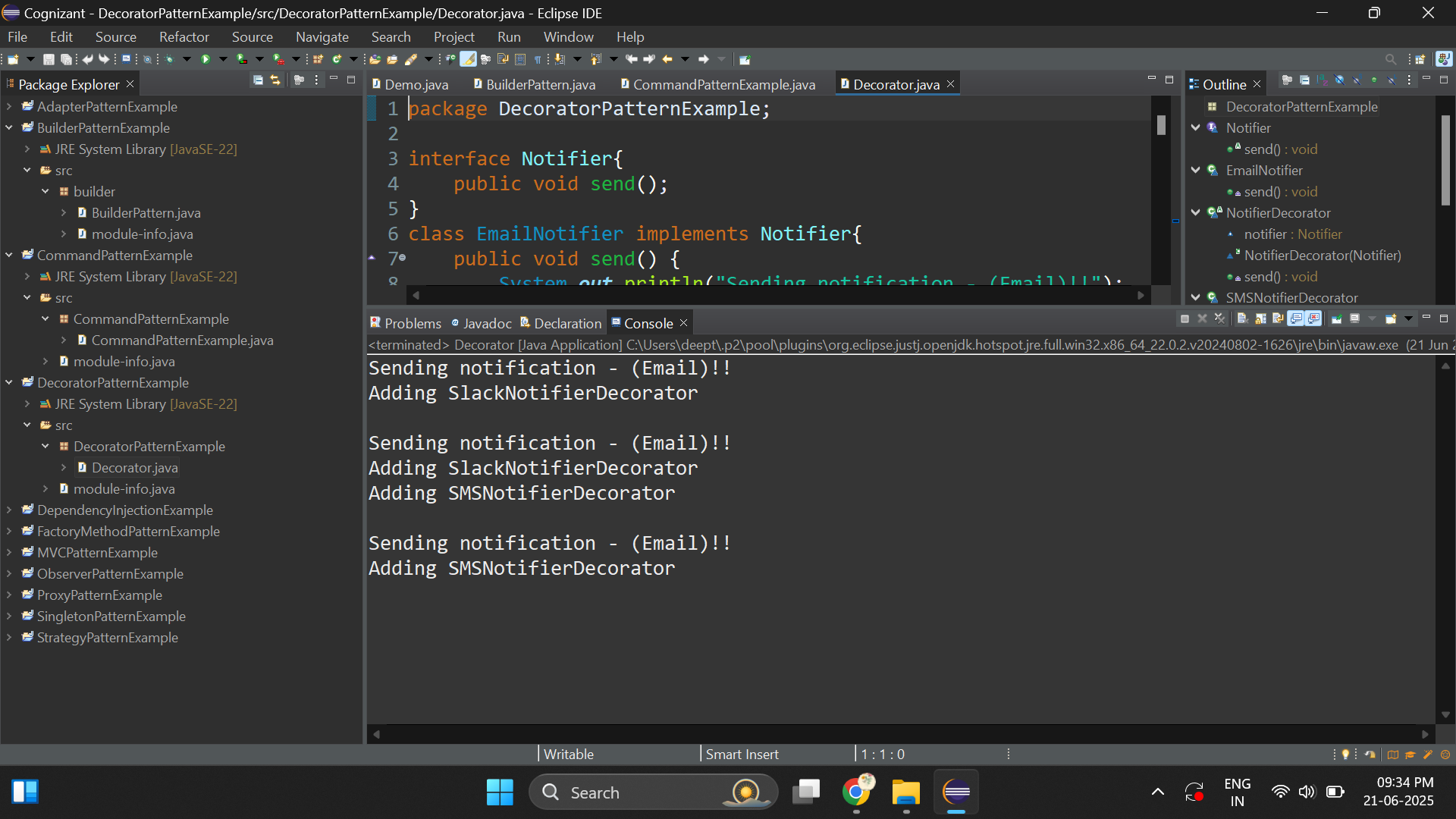
notify2=new SMSNotifierDecorator(notify2);

notify2.send();

}

}

**Output:**



**Exercise 6: Implementing the Proxy Pattern**

package \_6\_ProxyPatternExample;

interface Image{

public void display();

}

//confidential data

class RealImage implements Image {

private String ImageName;

RealImage(String ImageName){

this.ImageName=ImageName;

LoadDisk(ImageName);

}

public String getImageName() {

return ImageName;

}

public void LoadDisk(String ImageName) {

System.***out***.println("Loading From Disk :"+getImageName());

}

public void display() {

System.***out***.println("Displaying requested Image :"+getImageName());

}

}

class ProxyImage implements Image{

private String user\_role;

private String imageName;

private RealImage image;

ProxyImage(String user\_role,String imageName){

this.user\_role=user\_role;

this.imageName=imageName;

}

public void display() {

//lazy initialization

if(user\_role.equalsIgnoreCase("Admin")) {

if(image==null) {

image=new RealImage(imageName);

}

image.display();

}else {

System.***out***.println("Permission denied for "+user\_role);

}

}

}

public class ProxyPatternExample {

public static void main(String[] args) {

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

ProxyImage admin=new ProxyImage("admin","image\_1010");

admin.display();

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

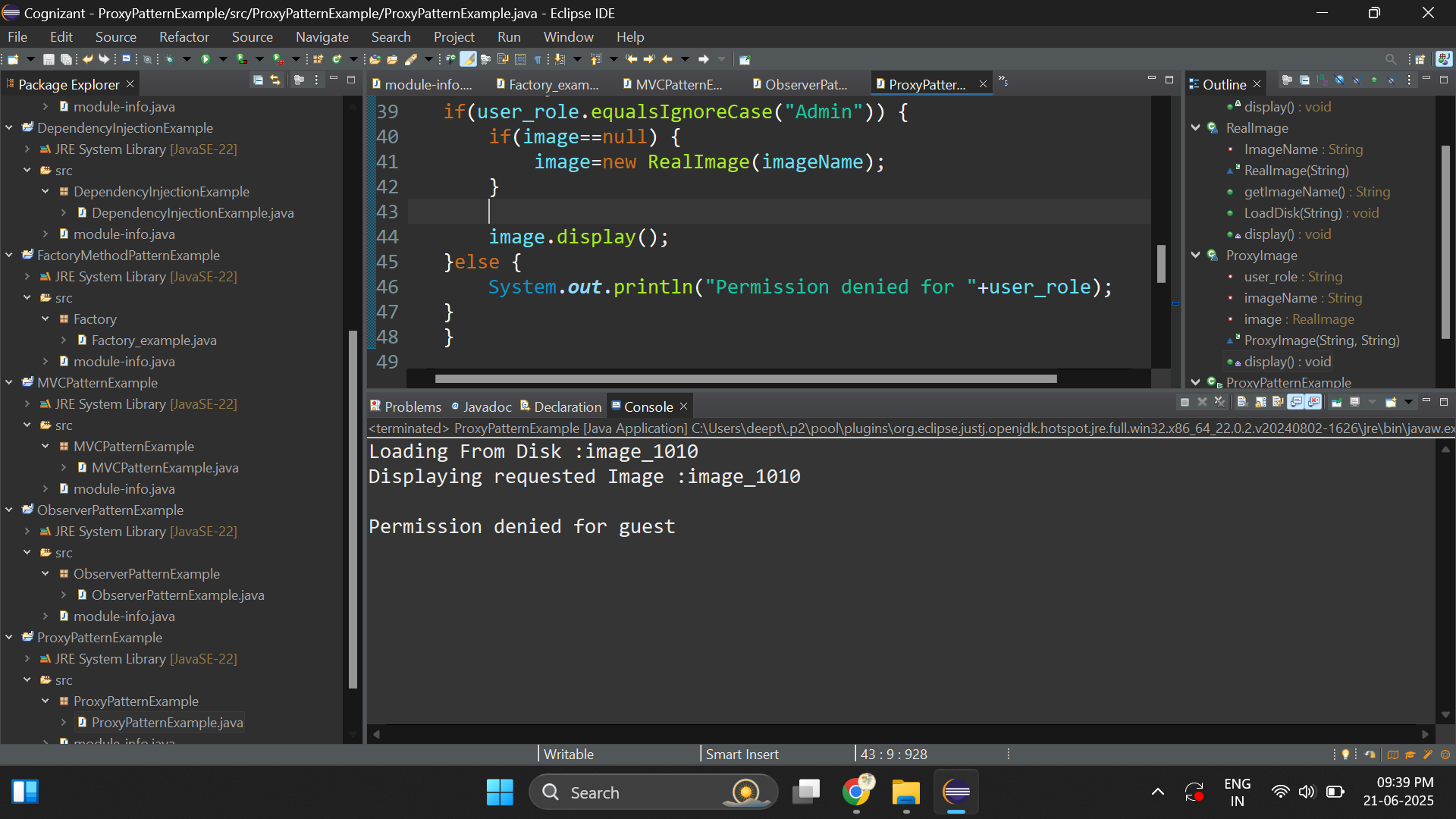
ProxyImage guest=new ProxyImage("guest","image\_1010");

guest.display();

}

}

**Output:**



**Exercise 7: Implementing the Observer Pattern**

package \_7\_ObserverPatternExample;

import java.util.\*;

//observer

interface Observer{

//multiple clients need to be notified whenever stock prices change

public void update(String stockprices);

}

class User implements Observer{

protected String name;

User(String name){

this.name=name;

}

public void update(String stockprices) {

System.***out***.println(name +" recevied notifiaction about stockprices -> " +stockprices);

}

}

//subject

interface Stock{

public void register(Observer obj);

public void deregister(Observer obj);

public void notify(String stockprices);

}

class StockMarket implements Stock{

//create a list to store the registered clients

ArrayList<Observer> observers=new ArrayList<>();

public void register(Observer obj) {

observers.add(obj);

}

public void deregister(Observer obj) {

observers.remove(obj);

}

public void notify(String stockprices) {

//send the notifications to all registered clients

for(Observer observer:observers) {

observer.update(stockprices);

}

}

}

public class ObserverPatternExample {

public static void main(String[] args) {

//create stockmarket

StockMarket stockmarket1=new StockMarket();

StockMarket stockmarket2=new StockMarket();

//create clients - stockmarket\_1

Observer user1=new User("Lisa");

Observer user2=new User("Abel");

//create clients - stockmarket\_2

Observer user3=new User("Deepthi");

Observer user4=new User("Ramakumar");

//add

stockmarket1.register(user1);

stockmarket1.register(user2);

stockmarket2.register(user3);

stockmarket2.register(user4);

stockmarket1.notify("Stock price is decreasing!");

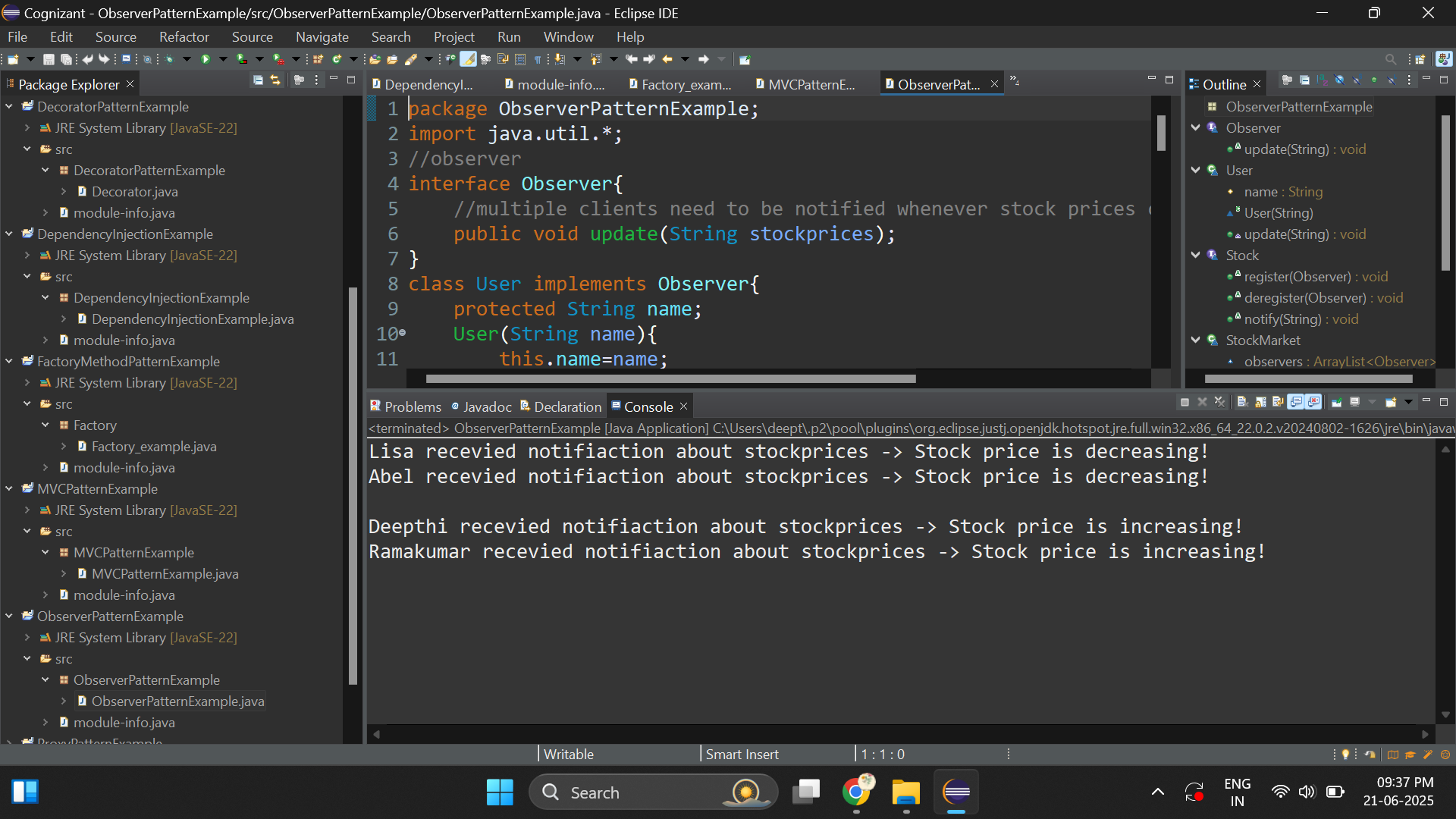
System.***out***.println();

stockmarket2.notify("Stock price is increasing!");

}

}

**Output:**



**Exercise 8: Implementing the Strategy Pattern**

package \_8\_StrategyPatternexample;

//Strategy

interface PaymentStrategy{

//Receiver name with amount

public void pay(String name,double amount);

}

class CreditCardPayment implements PaymentStrategy{

public void pay(String name,double amount) {

System.***out***.println("Payment method - CreditCardPayment ");

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

}

}

class PayPalPayment implements PaymentStrategy{

public void pay(String name,double amount) {

System.***out***.println("Payment method - PayPalPayment ");

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

}

}

//context class

class PaymentContext{

PaymentStrategy strategy;

//method - 1

public void setStrategy(PaymentStrategy strategy) {

this.strategy=strategy;

}

//method - 2

public void details(String name,double amount) {

strategy.pay(name,amount);

}

}

public class StrategyPatternExample {

public static void main(String[] args) {

PaymentContext context=new PaymentContext();

context.setStrategy(new CreditCardPayment());

context.details("Ramakumar",5000);

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

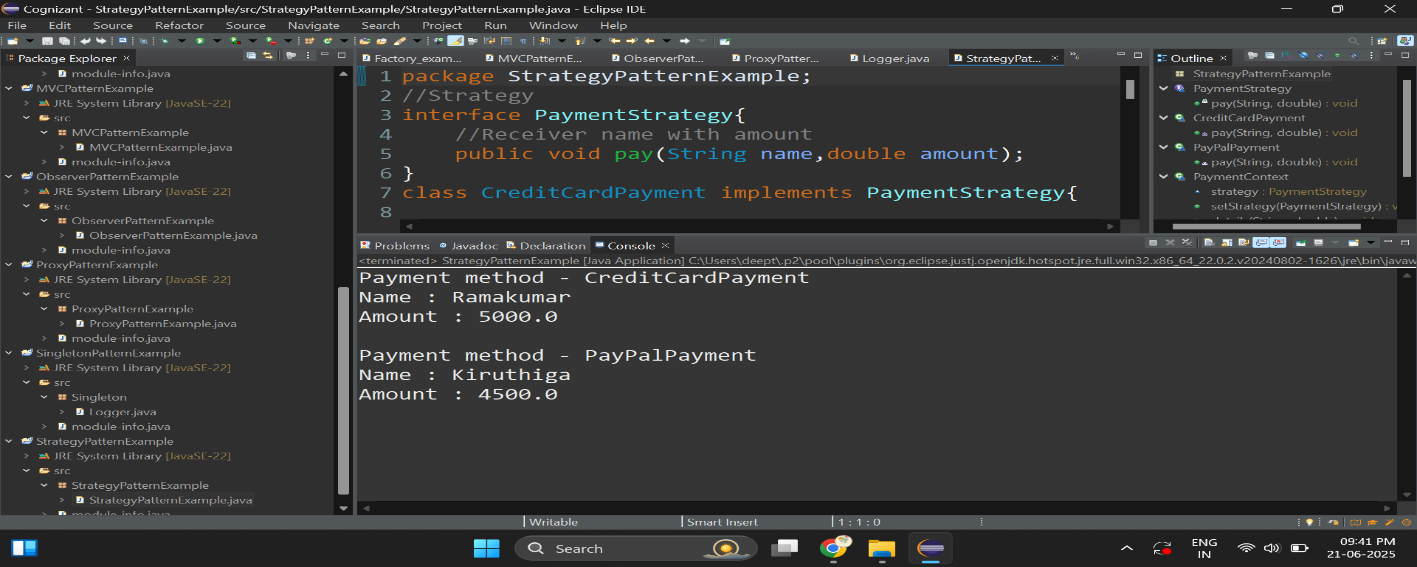
context.setStrategy(new PayPalPayment());

context.details("Kiruthiga",4500);

}

}

**Output:**



**Exercise 9: Implementing the Command Pattern**

package \_9\_CommandPatternExample;

import java.util.\*;

// Receiver

class Light {

public void turnOn() {

System.***out***.println("Light is ON");

}

public void turnOff() {

System.***out***.println("Light is OFF");

}

}

// Command Interface

interface Command {

void execute();

void undo(); // optional

}

// Concrete Command to turn on the light

class LightOnCommand implements Command {

Light light;

public LightOnCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOn();

}

public void undo() {

light.turnOff();

}

}

// Concrete Command to turn off the light

class LightOffCommand implements Command {

Light light;

public LightOffCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOff();

}

public void undo() {

light.turnOn();

}

}

// Invoker

class RemoteControl {

// Stack to store command history

Stack<Command> history = new Stack<>();

public void pressButton(Command command) {

command.execute();

history.push(command);

}

public void undoLast() {

if (!history.isEmpty()) {

Command lastCommand = history.pop();

System.***out***.print("Undoing the last command: ");

lastCommand.undo();

} else {

System.***out***.println("Nothing to undo. History is empty!");

}

}

// Execute a list of commands

public void queue(List<Command> commands) {

for (Command command : commands) {

pressButton(command);

}

}

}

// Client

public class CommandPatternExample {

public static void main(String[] args) {

Light light = new Light();

Command turnOn = new LightOnCommand(light);

Command turnOff = new LightOffCommand(light);

RemoteControl remote = new RemoteControl();

System.***out***.println("Executing step by step...");

remote.pressButton(turnOn);

remote.pressButton(turnOff);

remote.pressButton(turnOn);

remote.undoLast();

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

System.***out***.println("Executing Queue...");

List<Command> commandList = Arrays.*asList*(turnOn, turnOff, turnOff);

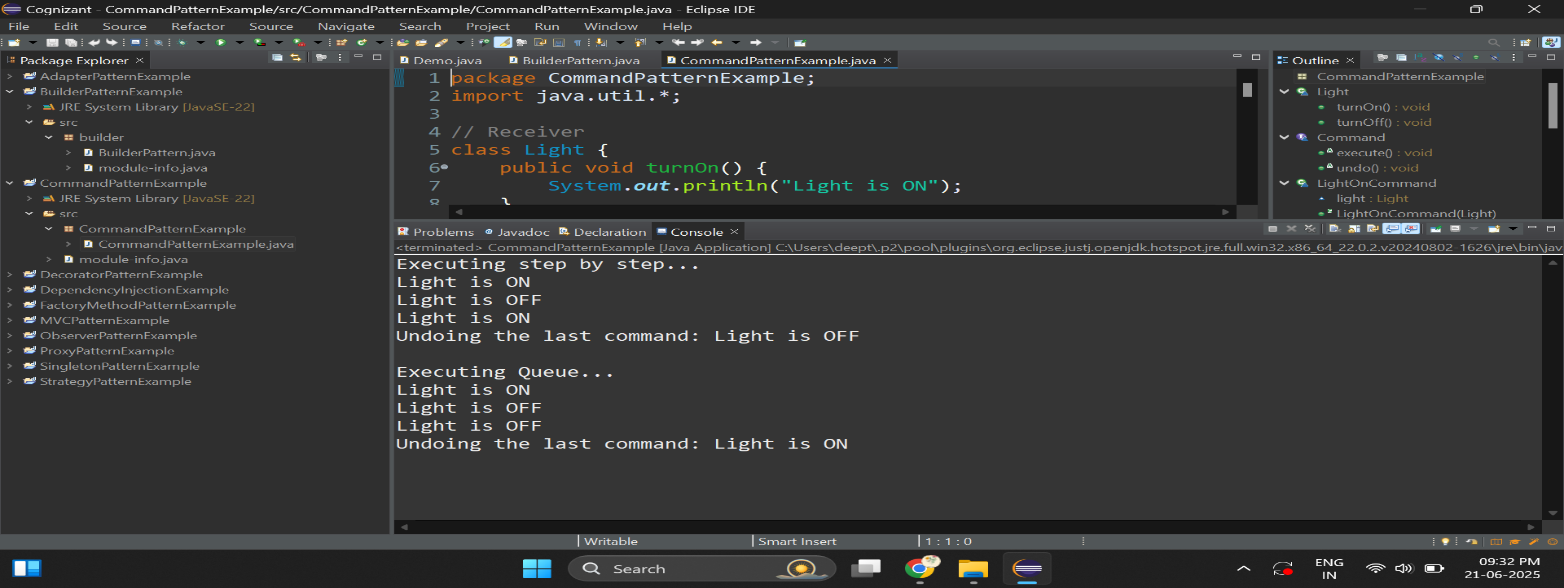
remote.queue(commandList);

remote.undoLast();

}

}

**Output:**



**Exercise 10: Implementing the MVC Pattern**

package \_10\_MVCPatternExample;

//Model - Data Layer -> Fetch the data from the database

class Student {

public String Name;

public int Id;

public char Grade;

public String getName() {

return Name;

}

public int getId() {

return Id;

}

public char getGrade() {

return Grade;

}

public void setName(String Name) {

Name=Name.trim();

this.Name=Name;

}

public void setId(int Id) {

this.Id=Id;

}

public void setGrade(char Grade) {

this.Grade=Grade;

}

}

//View ->Converts the data into human readable format

class StudentView {

public void displayStudentDetails(String name,int id,char grade) {

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

System.***out***.println("Name:"+name +"\nId:"+id+"\nGrade:"+grade);

}

}

class StudentController{

Student model;

StudentView view;

StudentController(Student model,StudentView view){

this.model=model;

this.view=view;

}

public void setStudentName(String name) {

model.setName(name);

}

public void setStudentId(int id) {

model.setId(id);

}

public void setStudentGrade(char c) {

model.setGrade(c);

}

public String getStudentName() {

return model.getName();

}

public int getStudentId() {

return model.getId();

}

public int getStudentGrade() {

return model.getGrade();

}

public void view() {

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

}

}

public class MVCPatternExample {

public static void main(String[] args) {

//database

Student student=new Student();

student.setName("Deepthikha");

student.setId(101);

student.setGrade('O');

StudentView view =new StudentView();

//send the request to model

//fetch the data ,then send to view

StudentController controller=new StudentController(student, view);

//update

controller.setStudentGrade('A');

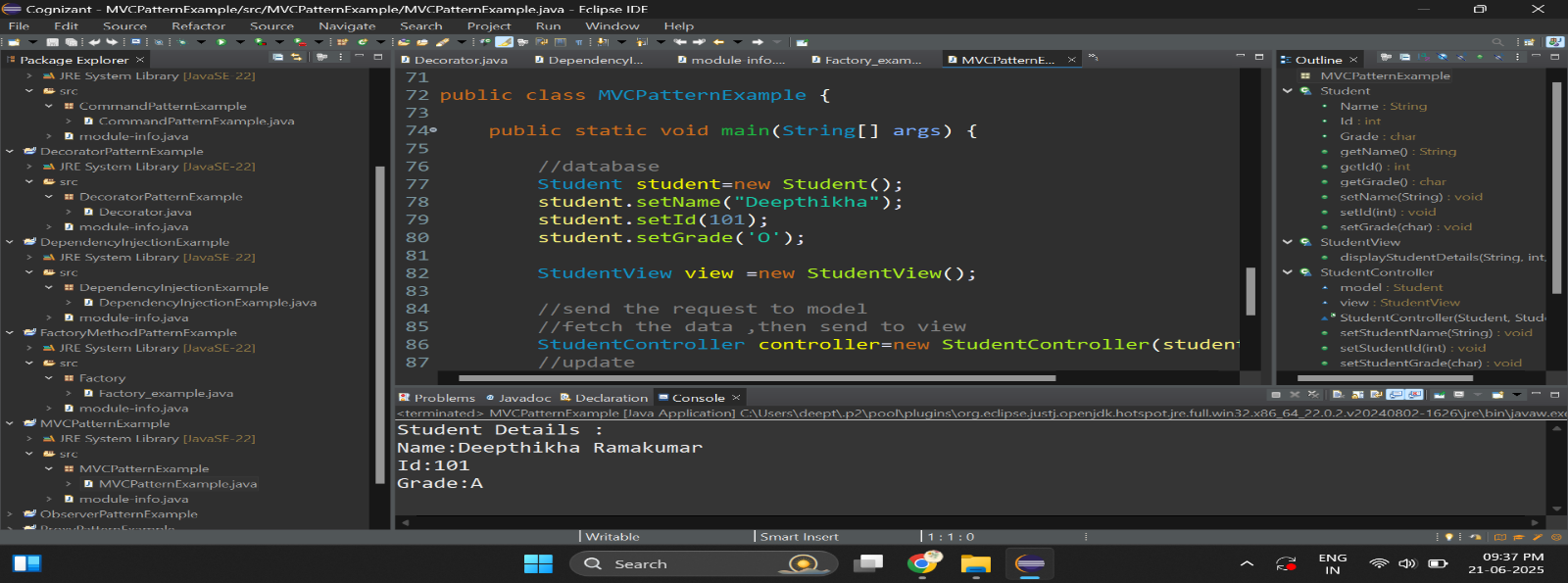
controller.setStudentName("Deepthikha Ramakumar");

controller.view();

}

}

**Output:**



**Exercise 11: Implementing Dependency Injection**

package DependencyPatternExample;

// Repository Interface

interface CustomerRepository {

void findCustomerById(String id);

}

// DB Repository Implementation

class CustomerRepositoryImpl implements CustomerRepository {

public void findCustomerById(String id) {

System.***out***.println("Fetching customer from Database with ID: " + id);

}

}

// In-Memory Repository Implementation

class InMemoryCustomerRepository implements CustomerRepository {

public void findCustomerById(String id) {

System.***out***.println("Fetching customer from In-Memory store with ID: " + id);

}

}

// Service Class

class CustomerService {

private CustomerRepository repository;

public CustomerService(CustomerRepository repository) {

this.repository = repository;

}

public void getCustomerDetails(String customerId) {

System.***out***.println("Customer Service: Getting details for ID = " + customerId);

repository.findCustomerById(customerId);

}

}

// Main Class

public class DependencyInjectionExample {

public static void main(String[] args) {

//database repo

CustomerRepository dbRepo = new CustomerRepositoryImpl();

CustomerService service = new CustomerService(dbRepo);

service.getCustomerDetails("1001");

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

CustomerRepository mRepo=new InMemoryCustomerRepository();

service=new CustomerService(mRepo);

service.getCustomerDetails("103564");

}

}

**Output:**

