**Cognizant\_Digital Nurture 4.0\_Deep Skilling**

**Engineering concepts**

**Module 1 - Design Patterns and Principles**

**Exercise 1**

**Project Name: SingletonPatternExample**

**Package: singleton**

**Logger.java**

package singleton;

public class Logger {

private static Logger instance;

private Logger() {

System.out.println("Logger Initialized");

}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger(); // only created once

}

return instance;

}

public void log(String message) {

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

}

}

**LoggerTest.java**

package singleton;

public class LoggerTest {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("This is the first log message.");

Logger logger2 = Logger.getInstance();

logger2.log("This is the second log message.");

if (logger1 == logger2) {

System.out.println("Both logger instances are the same (Singleton works!).");

} else {

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

}

}

}

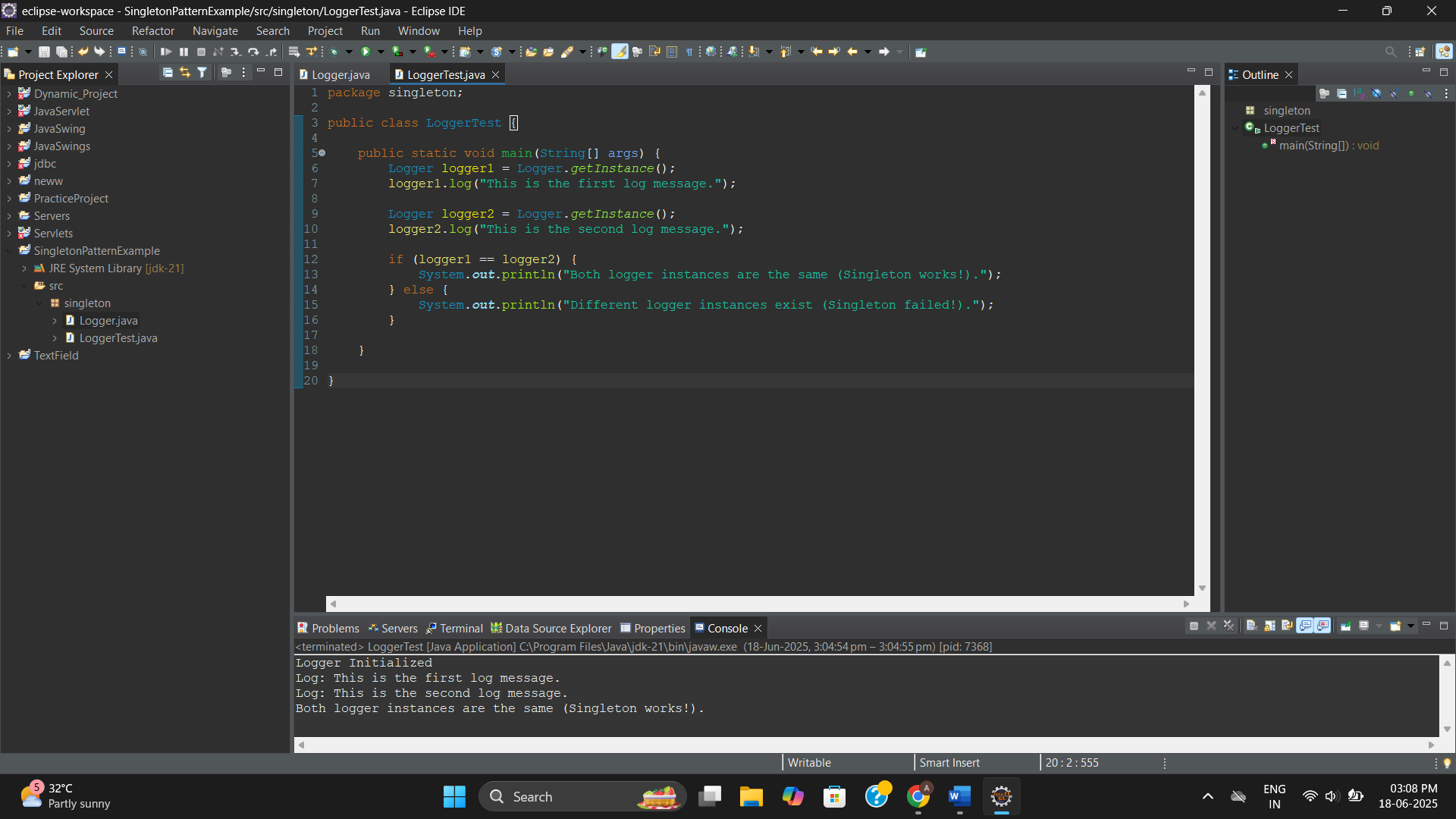
**Output:**

Logger Initialized

Log: This is the first log message.

Log: This is the second log message.

Both logger instances are the same (Singleton works!).

****

**Exercise 2**

**Project Name: FactoryMethodPatternExample**

**Package: factory**

**Document.java**

package factory;

public interface Document {

void open();

}

**WordDocument.java**

package factory;

public class WordDocument implements Document {

@Override

public void open() {

System.out.println("Opening a Word document.");

}

}

**PdfDocument.java**

package factory;

public class PdfDocument implements Document {

@Override

public void open() {

System.out.println("Opening a PDF document.");

}

}

**ExcelDocument.java**

package factory;

public class ExcelDocument implements Document {

@Override

public void open() {

System.out.println("Opening an Excel document.");

}

}

**DocumentFactory.java**

package factory;

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordDocumentFactory.java**

package factory;

public class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java**

package factory;

public class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**

package factory;

public class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

**FactoryTest.java**

package factory;

public class FactoryTest {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

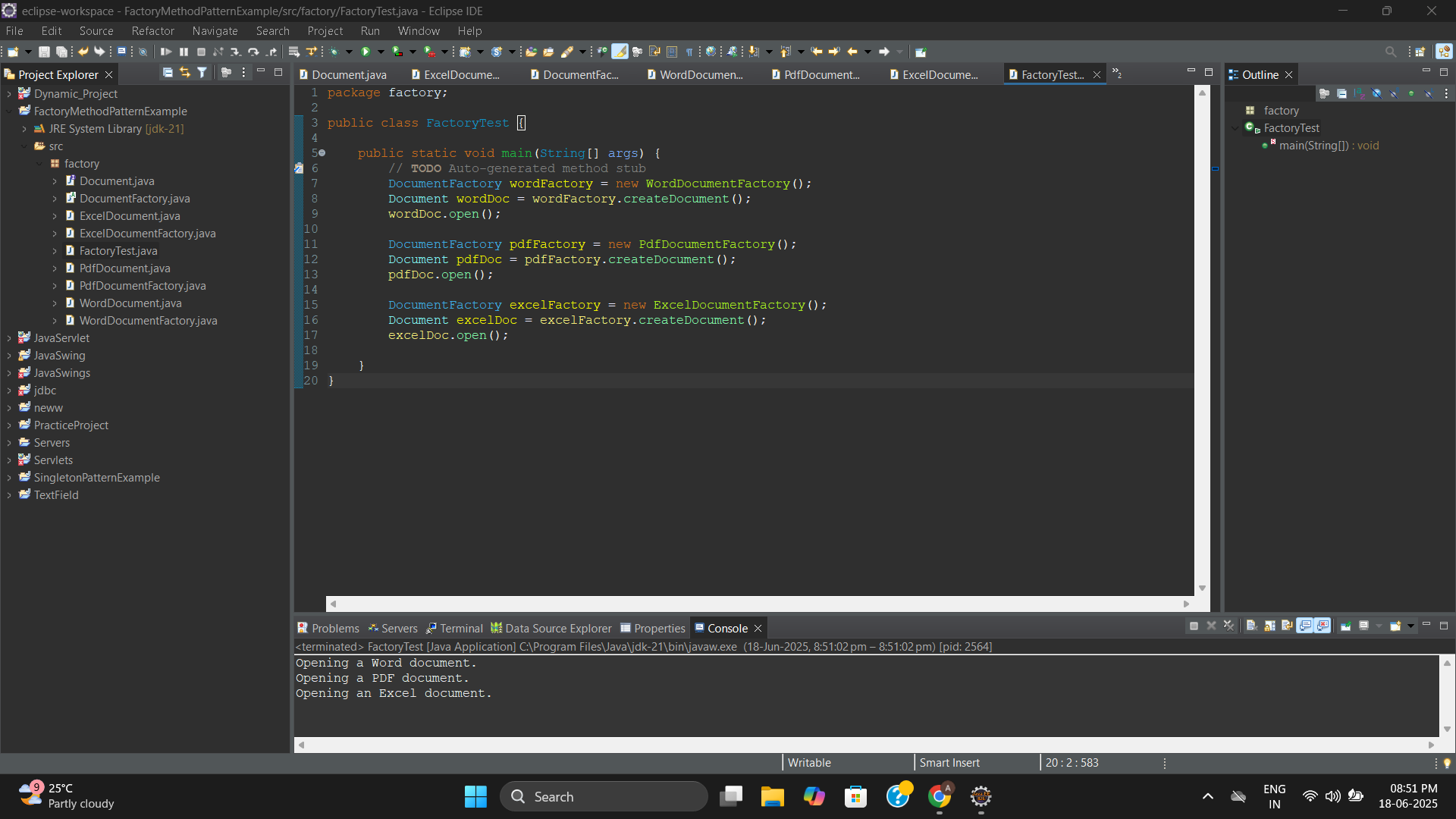
}

**Output:**

Opening a Word document.

Opening a PDF document.

Opening an Excel document.

****

**Exercise 3**

**Project Name: BuilderPatternExample**

**Package: builder**

**Computer.java**

package builder;

public class Computer {

private final String cpu;

private final String ram;

private final String storage;

private final String graphicsCard;

private Computer(Builder builder) {

this.cpu = builder.cpu;

this.ram = builder.ram;

this.storage = builder.storage;

this.graphicsCard = builder.graphicsCard;

}

public String getCpu() { return cpu; }

public String getRam() { return ram; }

public String getStorage() { return storage; }

public String getGraphicsCard() { return graphicsCard; }

public void showConfig() {

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

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

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

System.out.println("Graphics Card: " + graphicsCard);

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

}

public static class Builder {

private final String cpu;

private final 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);

}

}

}

**BuilderTest.java**

package builder;

public class BuilderTest {

public static void main(String[] args) {

// TODO Auto-generated method stub

Computer basicComputer = new Computer.Builder("Intel i5", "8GB")

.build();

basicComputer.showConfig();

Computer gamingComputer = new Computer.Builder("AMD Ryzen 9", "32GB")

.setStorage("1TB SSD")

.setGraphicsCard("NVIDIA RTX 4080")

.build();

gamingComputer.showConfig();

Computer officeComputer = new Computer.Builder("Intel i3", "4GB")

.setStorage("256GB SSD")

.build();

officeComputer.showConfig();

}

}

**Output:**

CPU: Intel i5

RAM: 8GB

Storage: null

Graphics Card: null

---------------------------

CPU: AMD Ryzen 9

RAM: 32GB

Storage: 1TB SSD

Graphics Card: NVIDIA RTX 4080

---------------------------

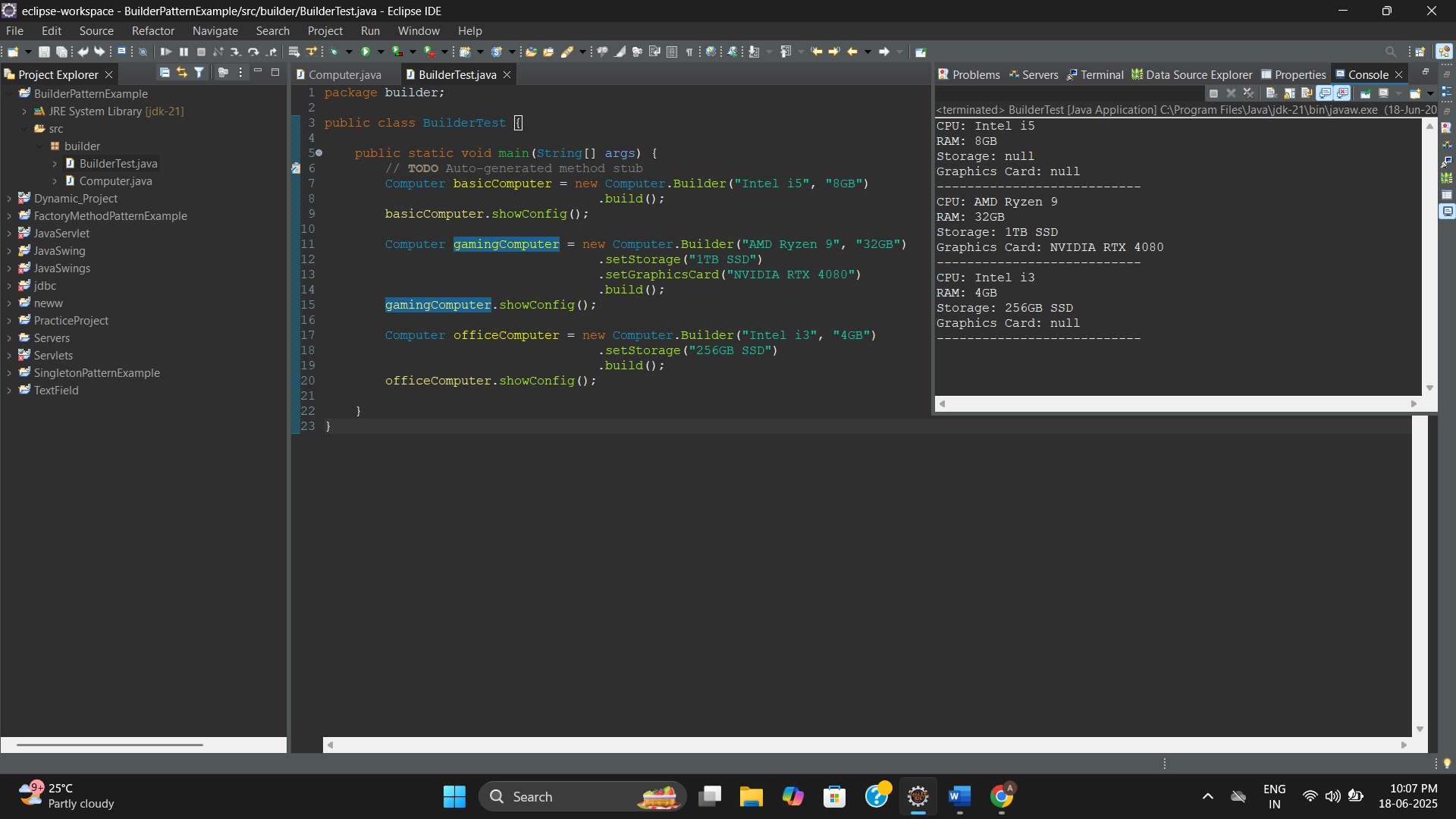
CPU: Intel i3

RAM: 4GB

Storage: 256GB SSD

Graphics Card: null

---------------------------

****

**Exercise 4**

**Project Name: AdapterPatternExample**

**Package: adapter**

**PaymentProcessor.java**

package adapter;

public interface PaymentProcessor {

void processPayment(double amount);

}

**StripeGateway.java**

package adapter;

public class StripeGateway {

public void makeStripePayment(double amount) {

System.out.println("Processing payment of ₹" + amount + " through Stripe.");

}

}

**PaypalGateway.java**

package adapter;

public class PaypalGateway {

public void sendPayment(double amountInDollars) {

System.out.println("Processing payment of $" + amountInDollars + " through PayPal.");

}

}

**StripeAdapter.java**

package adapter;

public class StripeAdapter implements PaymentProcessor {

private StripeGateway stripe;

public StripeAdapter(StripeGateway stripe) {

this.stripe = stripe;

}

@Override

public void processPayment(double amount) {

stripe.makeStripePayment(amount);

}

}

**PaypalAdapter.java**

package adapter;

public class PaypalAdapter implements PaymentProcessor {

private PaypalGateway paypal;

public PaypalAdapter(PaypalGateway paypal) {

this.paypal = paypal;

}

@Override

public void processPayment(double amount) {

// Convert INR to USD (assuming 1 USD = 83 INR for this example)

double amountInDollars = amount / 83;

paypal.sendPayment(amountInDollars);

}

}

**PaymentTest.java**

package adapter;

public class PaypalAdapter implements PaymentProcessor {

private PaypalGateway paypal;

public PaypalAdapter(PaypalGateway paypal) {

this.paypal = paypal;

}

@Override

public void processPayment(double amount) {

// Convert INR to USD (assuming 1 USD = 83 INR for this example)

double amountInDollars = amount / 83;

paypal.sendPayment(amountInDollars);

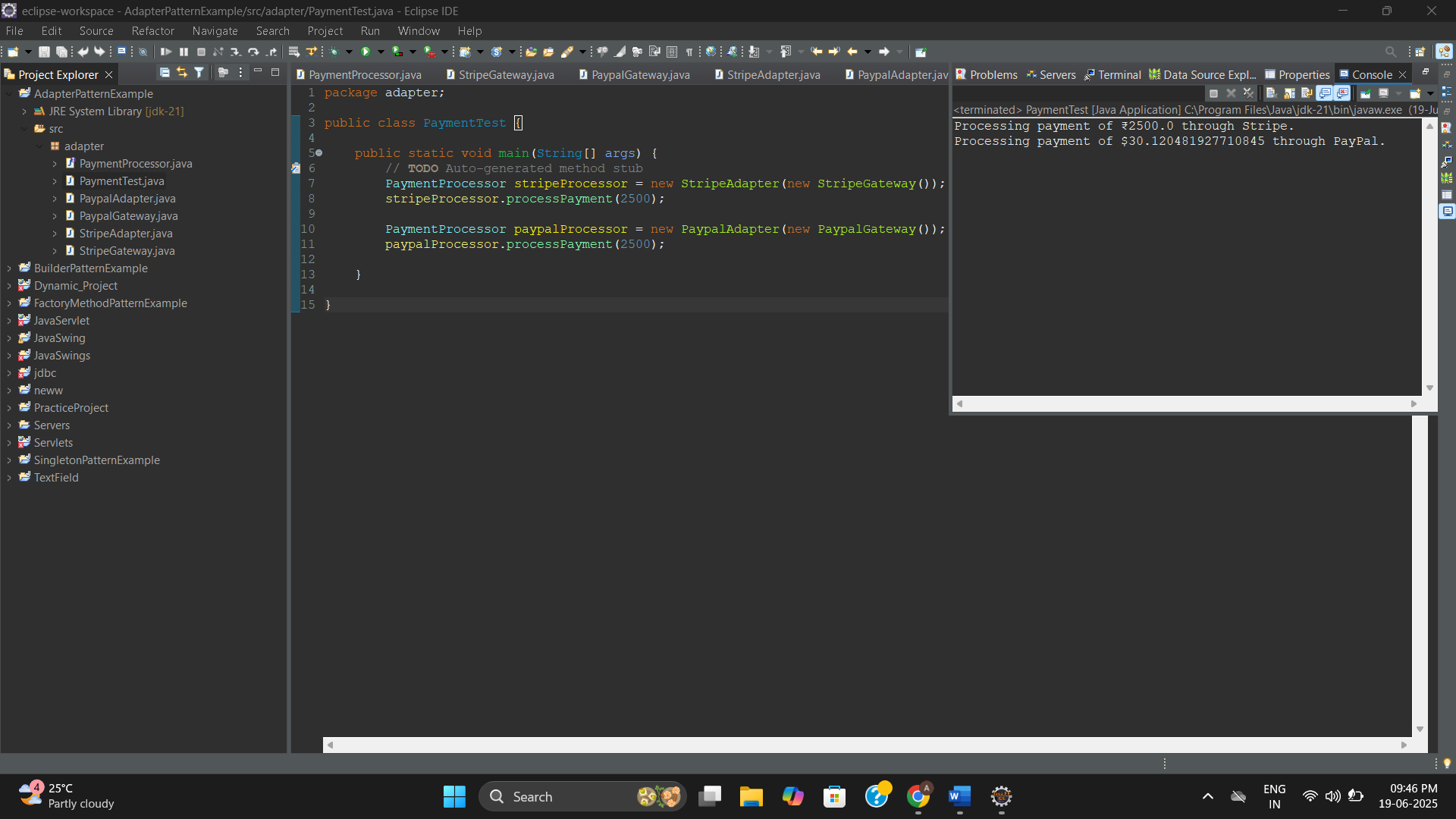
}

}

**Output:**

Processing payment of ₹2500.0 through Stripe.

Processing payment of $30.120481927710843 through PayPal.

****

**Exercise 5**

**Project Name: DecoratorPatternExample**

**Package: decorator**

**Notifier.java**

package decorator;

public interface Notifier {

void send(String message);

}

**EmailNotifier.java**

package decorator;

public class EmailNotifier implements Notifier {

@Override

public void send(String message) {

System.out.println("Sending Email: " + message);

}

}

**NotifierDecorator.java**

package decorator;

public abstract class NotifierDecorator implements Notifier {

protected Notifier wrappee;

public NotifierDecorator(Notifier notifier) {

this.wrappee = notifier;

}

@Override

public void send(String message) {

wrappee.send(message);

}

}

**SMSNotifierDecorator.java**

package decorator;

public class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

sendSMS(message);

}

private void sendSMS(String message) {

System.out.println("Sending SMS: " + message);

}

}

**SlackNotifierDecorator.java**

package decorator;

public class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

sendSlack(message);

}

private void sendSlack(String message) {

System.out.println("Sending Slack message: " + message);

}

}

**DecoratorTest.java**

package decorator;

public class DecoratorTest {

public static void main(String[] args) {

// Send via Email only

Notifier emailOnly = new EmailNotifier();

emailOnly.send("System Alert: Email Only");

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

// Send via Email + SMS

Notifier emailAndSMS = new SMSNotifierDecorator(new EmailNotifier());

emailAndSMS.send("System Alert: Email + SMS");

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

// Send via Email + SMS + Slack

Notifier allChannels = new SlackNotifierDecorator(

new SMSNotifierDecorator(

new EmailNotifier()

)

);

allChannels.send("System Alert: Email + SMS + Slack");

}

}

**Output:**

Sending Email: System Alert: Email Only

-------------------------

Sending Email: System Alert: Email + SMS

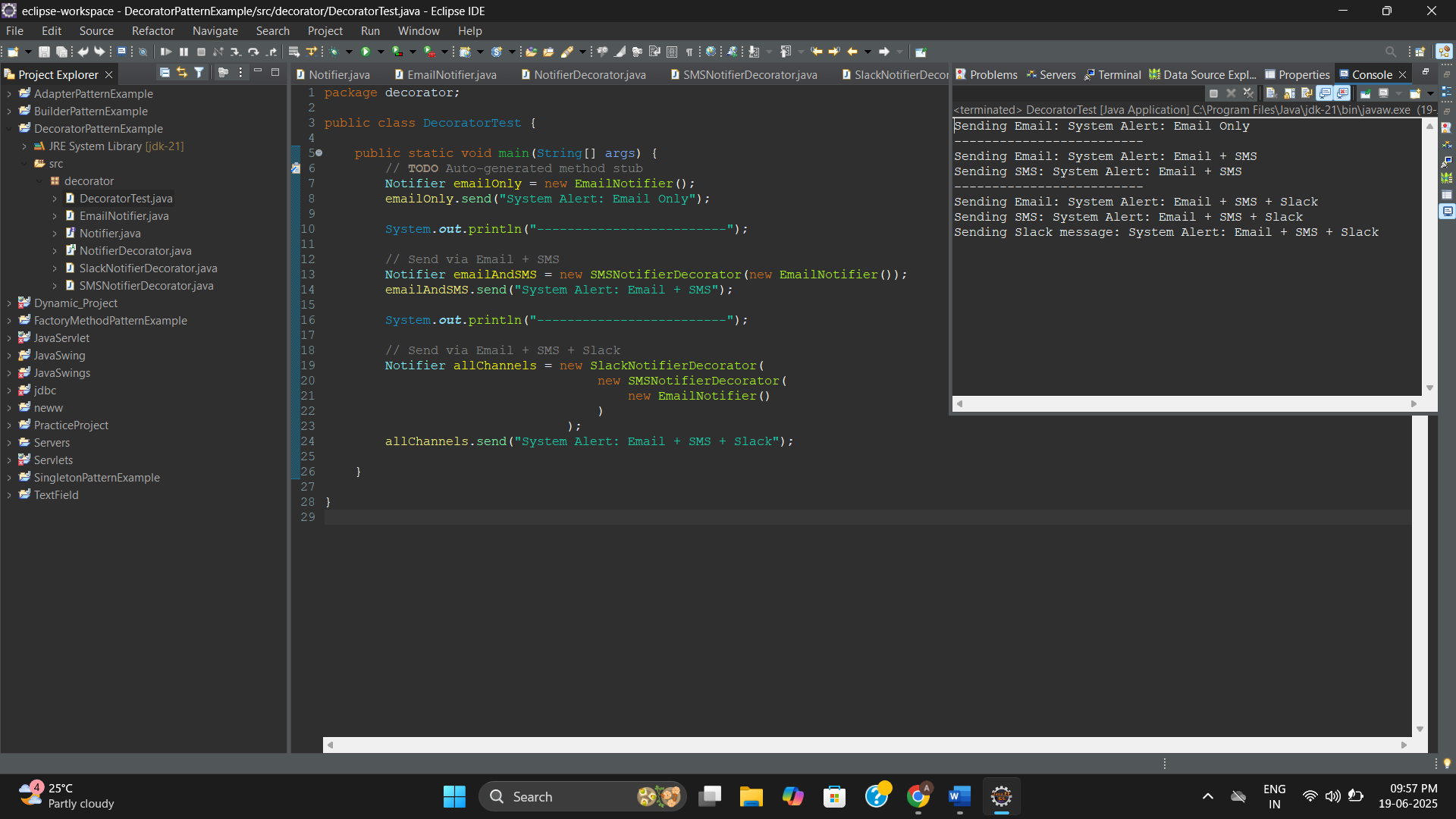
Sending SMS: System Alert: Email + SMS

-------------------------

Sending Email: System Alert: Email + SMS + Slack

Sending SMS: System Alert: Email + SMS + Slack

Sending Slack message: System Alert: Email + SMS + Slack

****

**Exercise 6**

**Project Name: ProxyPatternExample**

**Package: proxy**

**Image.java**

package proxy;

public interface Image {

void display();

}

**RealImage.java**

package proxy;

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);

}

}

**ProxyImage.java**

package proxy;

public class ProxyImage implements Image {

private String filename;

private RealImage realImage;

public ProxyImage(String filename) {

this.filename = filename;

}

@Override

public void display() {

if (realImage == null) {

realImage = new RealImage(filename);

} else {

System.out.println("Using cached image: " + filename);

}

realImage.display();

}

}

**ProxyTest.java**

package proxy;

public class ProxyTest {

public static void main(String[] args) {

Image image1 = new ProxyImage("pic1.jpg");

Image image2 = new ProxyImage("pic2.jpg");

image1.display();

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

image1.display();

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

image2.display();

}

}

**Output:**

Loading image from remote server: pic1.jpg

Displaying image: pic1.jpg

------------------------

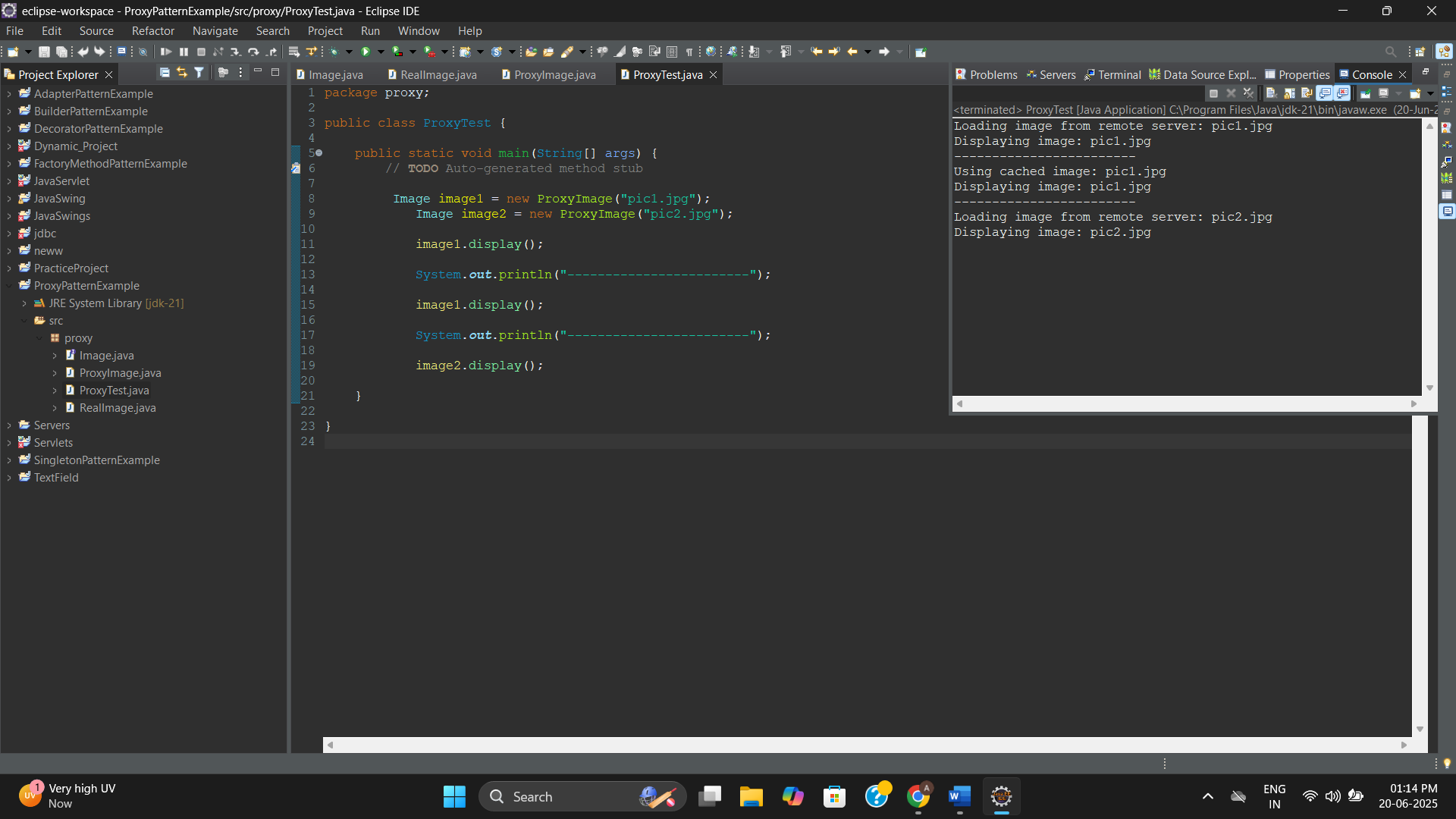
Using cached image: pic1.jpg

Displaying image: pic1.jpg

------------------------

Loading image from remote server: pic2.jpg

Displaying image: pic2.jpg

****

**Exercise 7**

**Project Name: ObserverPatternExample**

**Package: observer**

**Observer.java**

package observer;

public interface Observer {

void update(String stockName, double price);

}

**Stock.java**

package observer;

public interface Stock {

void registerObserver(Observer o);

void removeObserver(Observer o);

void notifyObservers(String stockName, double price);

}

**StockMarket.java**

package observer;

import java.util.ArrayList;

import java.util.List;

public class StockMarket implements Stock {

private List<Observer> observers;

public StockMarket() {

observers = new ArrayList<>();

}

@Override

public void registerObserver(Observer o) {

observers.add(o);

}

@Override

public void removeObserver(Observer o) {

observers.remove(o);

}

@Override

public void notifyObservers(String stockName, double price) {

for (Observer o : observers) {

o.update(stockName, price);

}

}

// Method to simulate stock price change

public void setStockPrice(String stockName, double price) {

System.out.println("Stock Updated: " + stockName + " = ₹" + price);

notifyObservers(stockName, price);

}

}

**MobileApp.java**

package observer;

public class MobileApp implements Observer {

private String appName;

public MobileApp(String appName) {

this.appName = appName;

}

@Override

public void update(String stockName, double price) {

System.out.println(appName + " Mobile App: " + stockName + " updated to ₹" + price);

}

}

**WebApp.java**

package observer;

public class WebApp implements Observer {

private String siteName;

public WebApp(String siteName) {

this.siteName = siteName;

}

@Override

public void update(String stockName, double price) {

System.out.println(siteName + " Web App: " + stockName + " updated to ₹" + price);

}

}

**ObserverTest.java**

package observer;

public class ObserverTest {

public static void main(String[] args) {

StockMarket market = new StockMarket();

Observer mobile1 = new MobileApp("InvestBuddy");

Observer web1 = new WebApp("StockTracker");

market.registerObserver(mobile1);

market.registerObserver(web1);

market.setStockPrice("TCS", 3550.00);

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

market.setStockPrice("INFY", 1480.25);

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

market.removeObserver(web1);

market.setStockPrice("WIPRO", 445.75);

}

}

**Output:**

Stock Updated: TCS = ₹3550.0

InvestBuddy Mobile App: TCS updated to ₹3550.0

StockTracker Web App: TCS updated to ₹3550.0

----------------------

Stock Updated: INFY = ₹1480.25

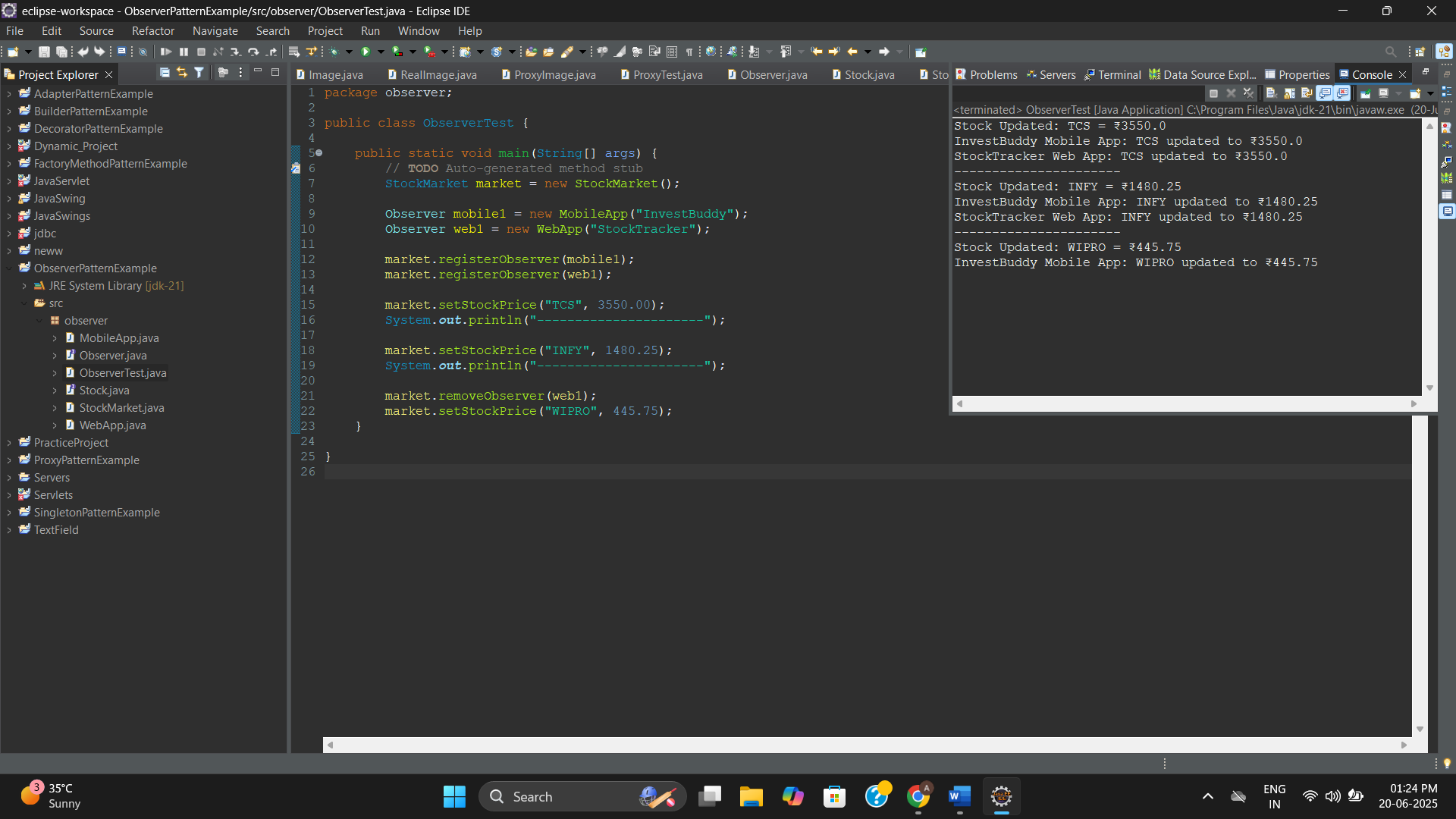
InvestBuddy Mobile App: INFY updated to ₹1480.25

StockTracker Web App: INFY updated to ₹1480.25

----------------------

Stock Updated: WIPRO = ₹445.75

InvestBuddy Mobile App: WIPRO updated to ₹445.75

****

**Exercise 8**

**Project Name: StrategyPatternExample**

**Package: strategy**

**PaymentStrategy.java**

package strategy;

public interface PaymentStrategy {

void pay(double amount);

}

**CreditCardPayment.java**

package strategy;

public class CreditCardPayment implements PaymentStrategy {

private String cardNumber;

private String cardHolder;

public CreditCardPayment(String cardNumber, String cardHolder) {

this.cardNumber = cardNumber;

this.cardHolder = cardHolder;

}

@Override

public void pay(double amount) {

System.out.println("Paid ₹" + amount + " using Credit Card (" + cardHolder + ")");

}

}

**PayPalPayment.java**

package strategy;

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 account (" + email + ")");

}

}

**PaymentContext.java**

package strategy;

public class PaymentContext {

private PaymentStrategy strategy;

public void setPaymentStrategy(PaymentStrategy strategy) {

this.strategy = strategy;

}

public void payAmount(double amount) {

if (strategy == null) {

System.out.println("Payment strategy not set!");

} else {

strategy.pay(amount);

}

}

}

**StrategyTest.java**

package strategy;

public class StrategyTest {

public static void main(String[] args) {

PaymentContext context = new PaymentContext();

context.setPaymentStrategy(new CreditCardPayment("1234-5678-9876-5432", "Ananth"));

context.payAmount(2500);

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

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

context.payAmount(1800);

}

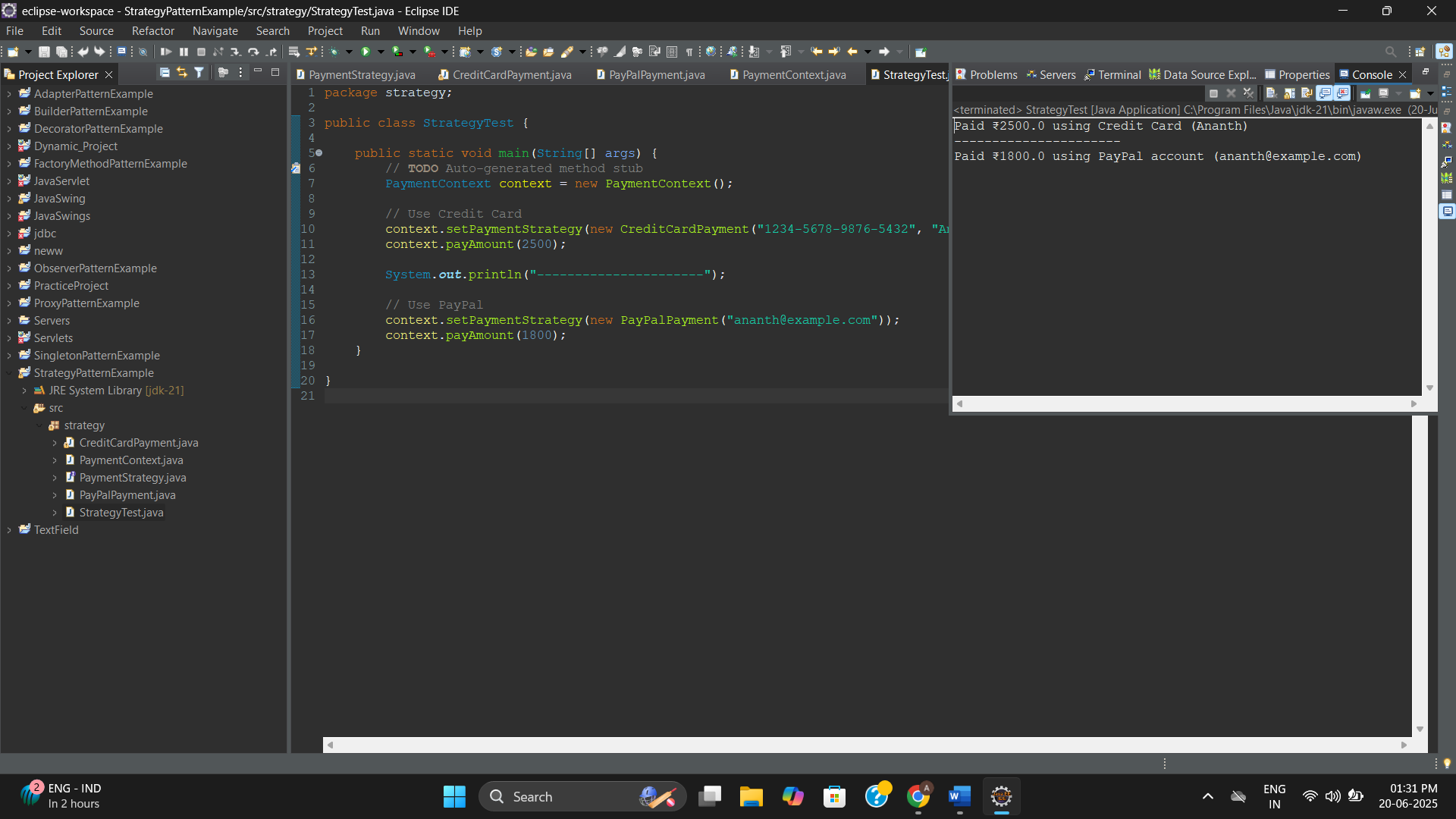
}

**Output:**

Paid ₹2500.0 using Credit Card (Ananth)

----------------------

Paid ₹1800.0 using PayPal account (ananth@example.com)

****

**Exercise 9**

**Project Name: CommandPatternExample**

**Package: command**

**Command.java**

package command;

public interface Command {

void execute();

}

**Light.java**

package command;

public class Light {

public void turnOn() {

System.out.println("The light is ON");

}

public void turnOff() {

System.out.println("The light is OFF");

}

}

**LightOnCommand.java**

package command;

public class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

@Override

public void execute() {

light.turnOn();

}

}

**LightOffCommand.java**

package command;

public class LightOffCommand implements Command {

private Light light;

public LightOffCommand(Light light) {

this.light = light;

}

@Override

public void execute() {

light.turnOff();

}

}

**RemoteControl.java**

package command;

public class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

if (command != null) {

command.execute();

} else {

System.out.println("No command set!");

}

}

}

**CommandTest.java**

package command;

public class CommandTest {

public static void main(String[] args) {

Light livingRoomLight = new Light();

Command lightOn = new LightOnCommand(livingRoomLight);

Command lightOff = new LightOffCommand(livingRoomLight);

RemoteControl remote = new RemoteControl();

remote.setCommand(lightOn);

remote.pressButton();

remote.setCommand(lightOff);

remote.pressButton();

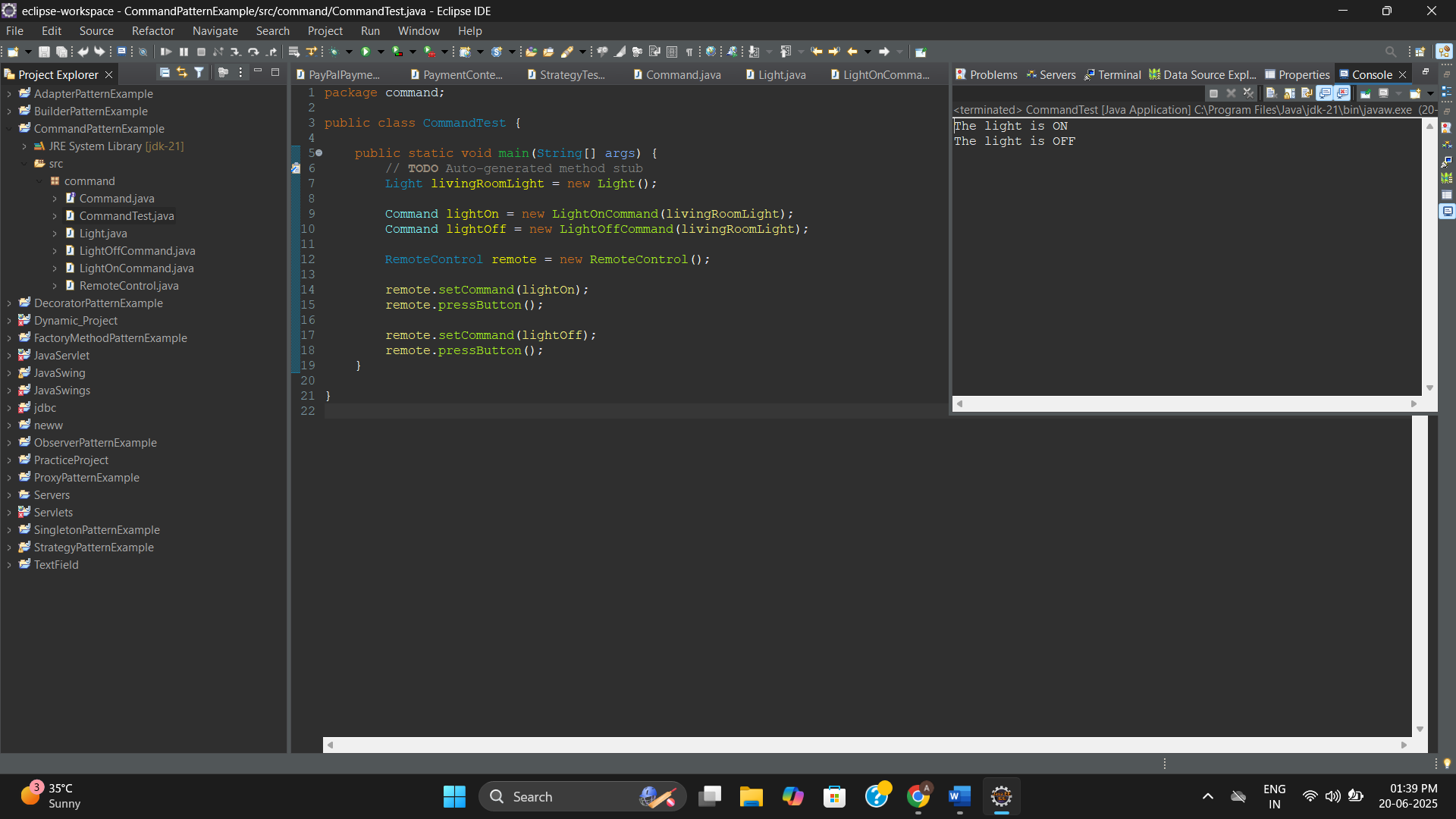
}

}

**Output:**

The light is ON

The light is OFF

****

**Exercise 10**

**Project Name: MVCPatternExample**

**Package: mvc**

**Student.java**

package mvc;

public class Student {

private String id;

private String name;

private String grade;

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

this.id = id;

this.name = name;

this.grade = grade;

}

// Getters and Setters

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getGrade() {

return grade;

}

public void setGrade(String grade) {

this.grade = grade;

}

}

**StudentView.java**

package mvc;

public class StudentView {

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

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

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

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

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

}

}

**StudentController.java**

package mvc;

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 void setStudentGrade(String grade) {

model.setGrade(grade);

}

public String getStudentName() {

return model.getName();

}

public String getStudentGrade() {

return model.getGrade();

}

public String getStudentId() {

return model.getId();

}

public void updateView() {

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

}

}

**MVCMain.java**

package mvc;

public class MVCMain {

public static void main(String[] args) {

Student student = new Student("EUEC017", "Ananth", "O");

StudentView view = new StudentView();

StudentController controller = new StudentController(student, view);

controller.updateView();

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

controller.setStudentName("Roopine");

controller.setStudentGrade("A+");

controller.updateView();

}

}

**Output:**

Student Details:

ID: EUEC017

Name: Ananth

Grade: O

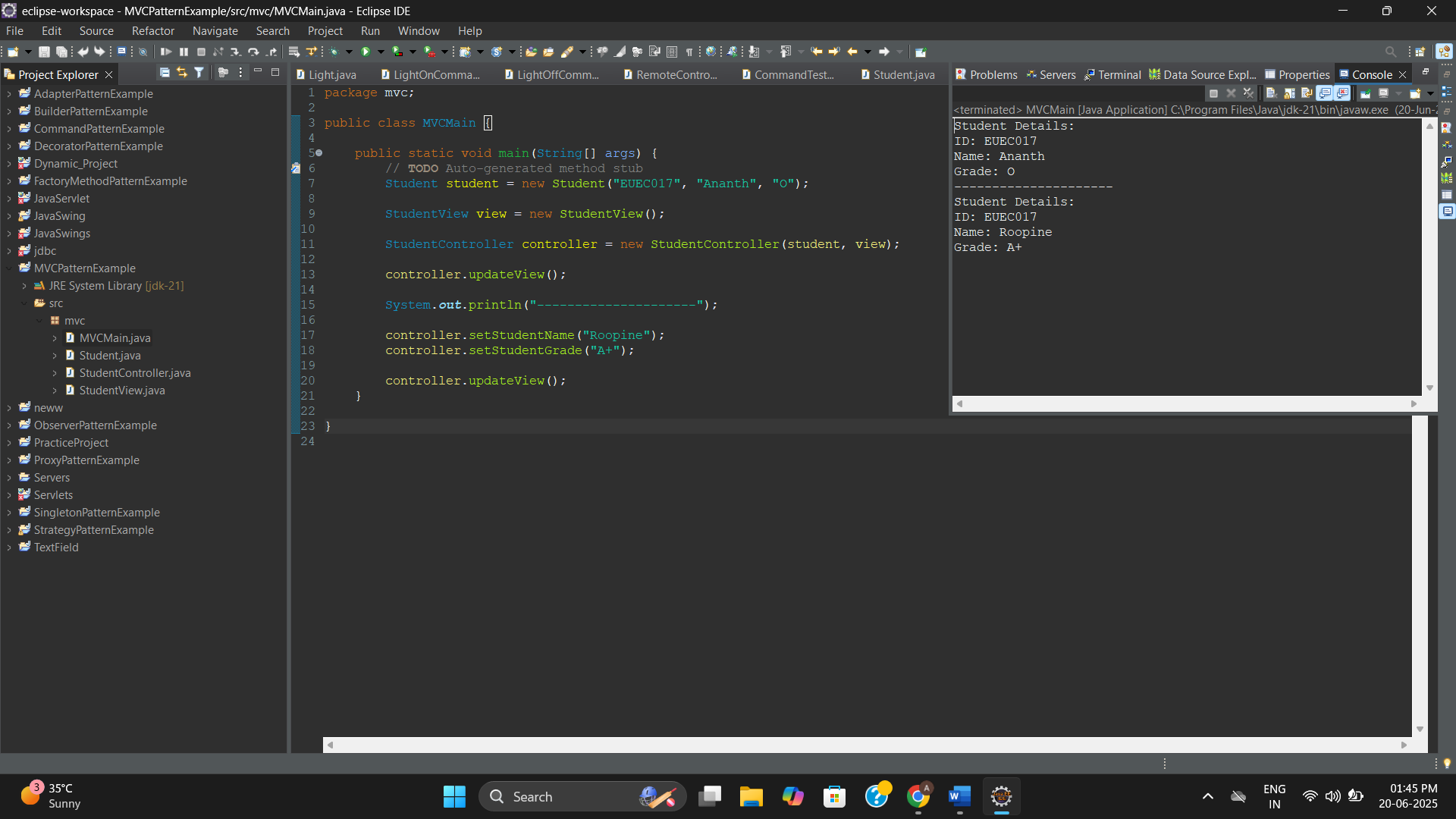
---------------------

Student Details:

ID: EUEC017

Name: Roopine

Grade: A+

****

**Exercise 11**

**Project Name: DependencyInjectionExample**

**Package: depend**

**Customer.java**

package depend;

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; }

}

**CustomerRepository.java**

package depend;

public interface CustomerRepository {

Customer findCustomerById(String id);

}

**CustomerRepositoryImpl.java**

package depend;

public class CustomerRepositoryImpl implements CustomerRepository {

@Override

public Customer findCustomerById(String id) {

return new Customer(id, "Customer\_" + id);

}

}

**CustomerService.java**

package depend;

public class CustomerService {

private final CustomerRepository repository;

public CustomerService(CustomerRepository repository) {

this.repository = repository;

}

public void displayCustomerDetails(String id) {

Customer customer = repository.findCustomerById(id);

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

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

}

}

**DIMain.java**

package depend;

public class DIMain {

public static void main(String[] args) {

CustomerRepository repository = new CustomerRepositoryImpl();

CustomerService service = new CustomerService(repository);

service.displayCustomerDetails("C101");

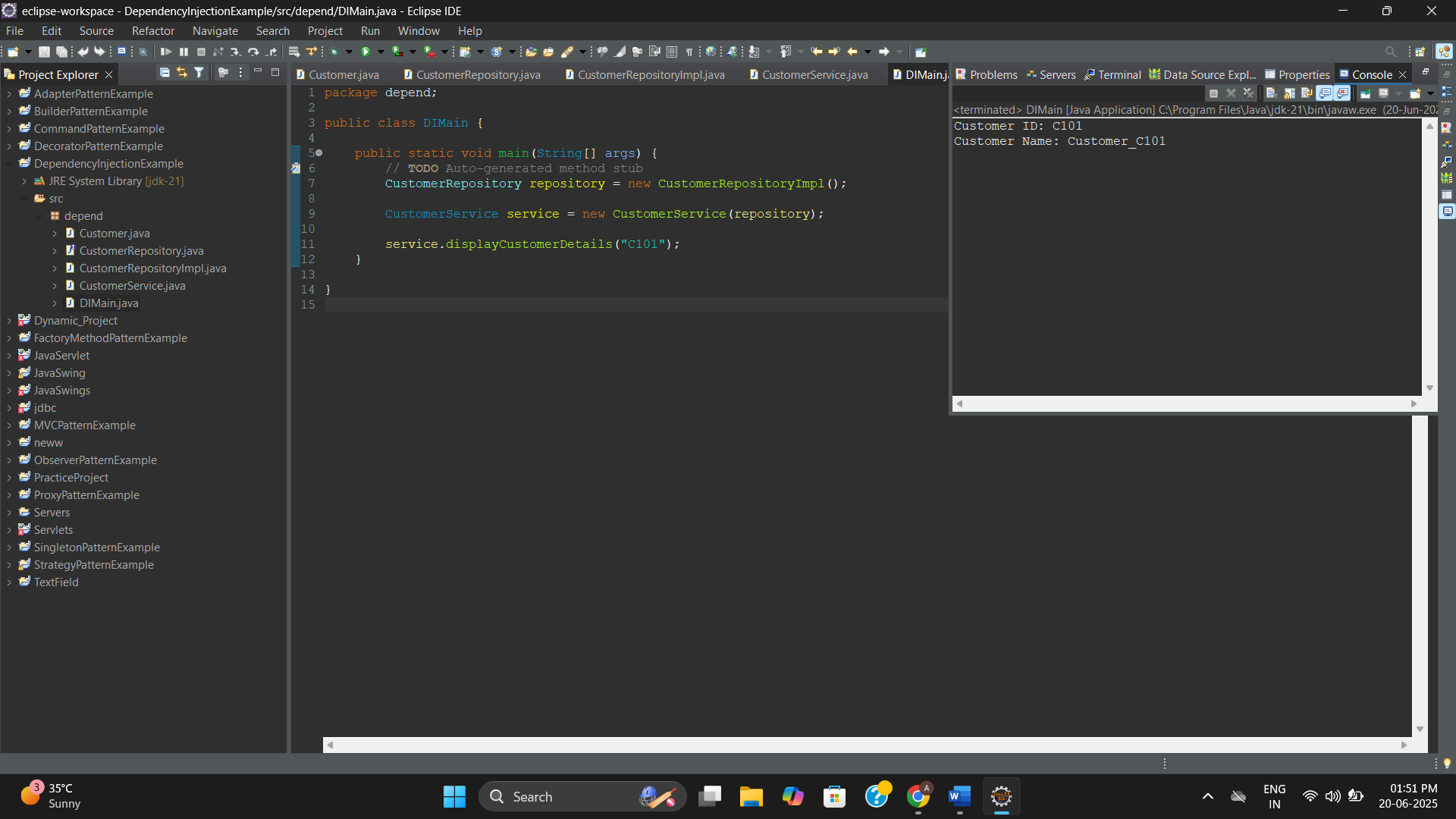
}

}

**Output:**

Customer ID: C101

Customer Name: Customer\_C101

****