**Exercise-1**

**Code:**

public class SingletonExample {

static class Logger {

private static Logger instance;

private Logger() {

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

}

public static Logger getInstance() {

return instance == null ? instance = new Logger() : instance;

}

public void log(String msg) {

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

}

}

public static void main(String[] args) {

Logger log1 = Logger.getInstance();

log1.log("App started");

Logger log2 = Logger.getInstance();

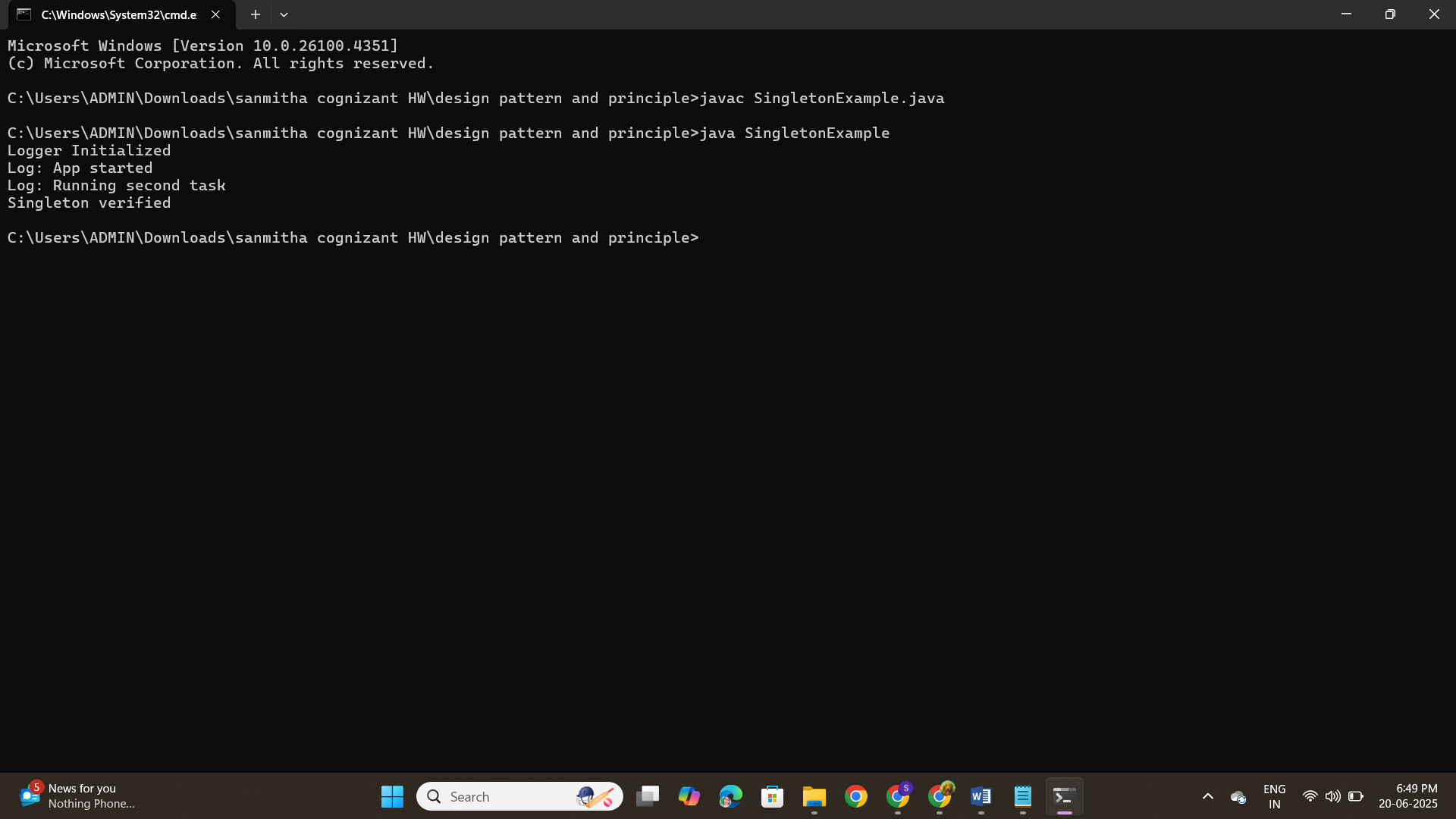
log2.log("Running second task");

System.out.println(log1 == log2 ? "Singleton verified" : "Singleton failed");

}

}

**Output:**



**EXERCISE-2**

**CODE:**

public class FactoryMethodExample {

// Document interface (common type)

interface Document {

void open();

}

// Concrete Document types

static class WordDocument implements Document {

public void open() {

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

}

}

static class PdfDocument implements Document {

public void open() {

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

}

}

static class ExcelDocument implements Document {

public void open() {

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

}

}

// Abstract Factory

static abstract class DocumentFactory {

abstract Document createDocument();

}

// Concrete Factories

static class WordDocumentFactory extends DocumentFactory {

Document createDocument() {

return new WordDocument();

}

}

static class PdfDocumentFactory extends DocumentFactory {

Document createDocument() {

return new PdfDocument();

}

}

static class ExcelDocumentFactory extends DocumentFactory {

Document createDocument() {

return new ExcelDocument();

}

}

// Test the Factory Method Pattern

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();

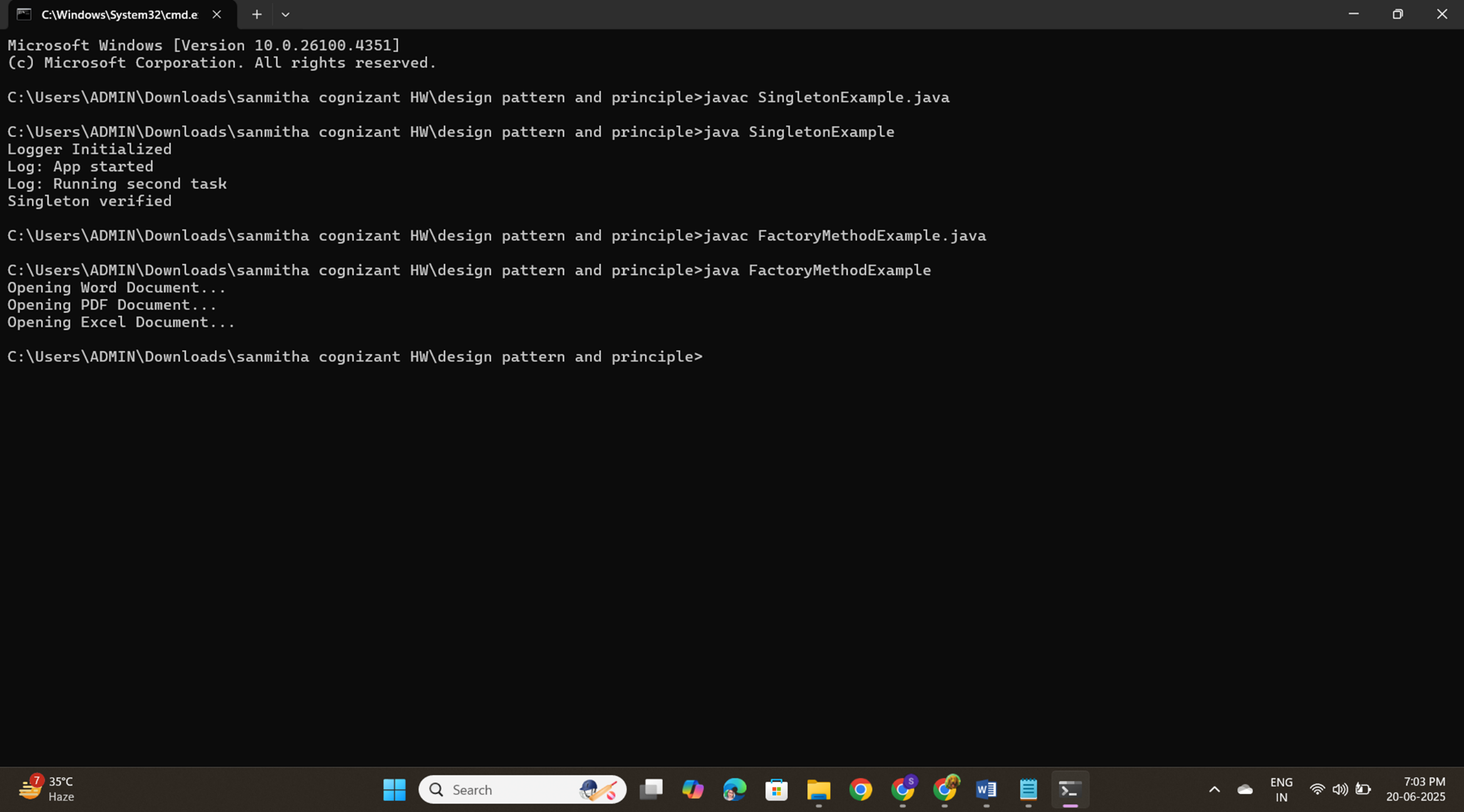
Document excel = excelFactory.createDocument();

excel.open();

}

}

**OUTPUT:**



**Exercise-3**

**CODE:**

public class BuilderPatternExample {

static 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.storage = builder.storage;

this.graphicsCard = builder.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("-------------------------");

}

static class Builder {

private String CPU;

private String RAM;

private String storage;

private String graphicsCard;

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 Builder setGraphicsCard(String graphicsCard) {

this.graphicsCard = graphicsCard;

return this;

}

public Computer build() {

return new Computer(this);

}

}

}

public static void main(String[] args) {

Computer basicPC = new Computer.Builder()

.setCPU("Intel i3")

.setRAM("8GB")

.setStorage("256GB SSD")

.build();

Computer gamingPC = new Computer.Builder()

.setCPU("Intel i9")

.setRAM("32GB")

.setStorage("1TB SSD")

.setGraphicsCard("NVIDIA RTX 4080")

.build();

System.out.println("Basic PC Configuration:");

basicPC.showConfig();

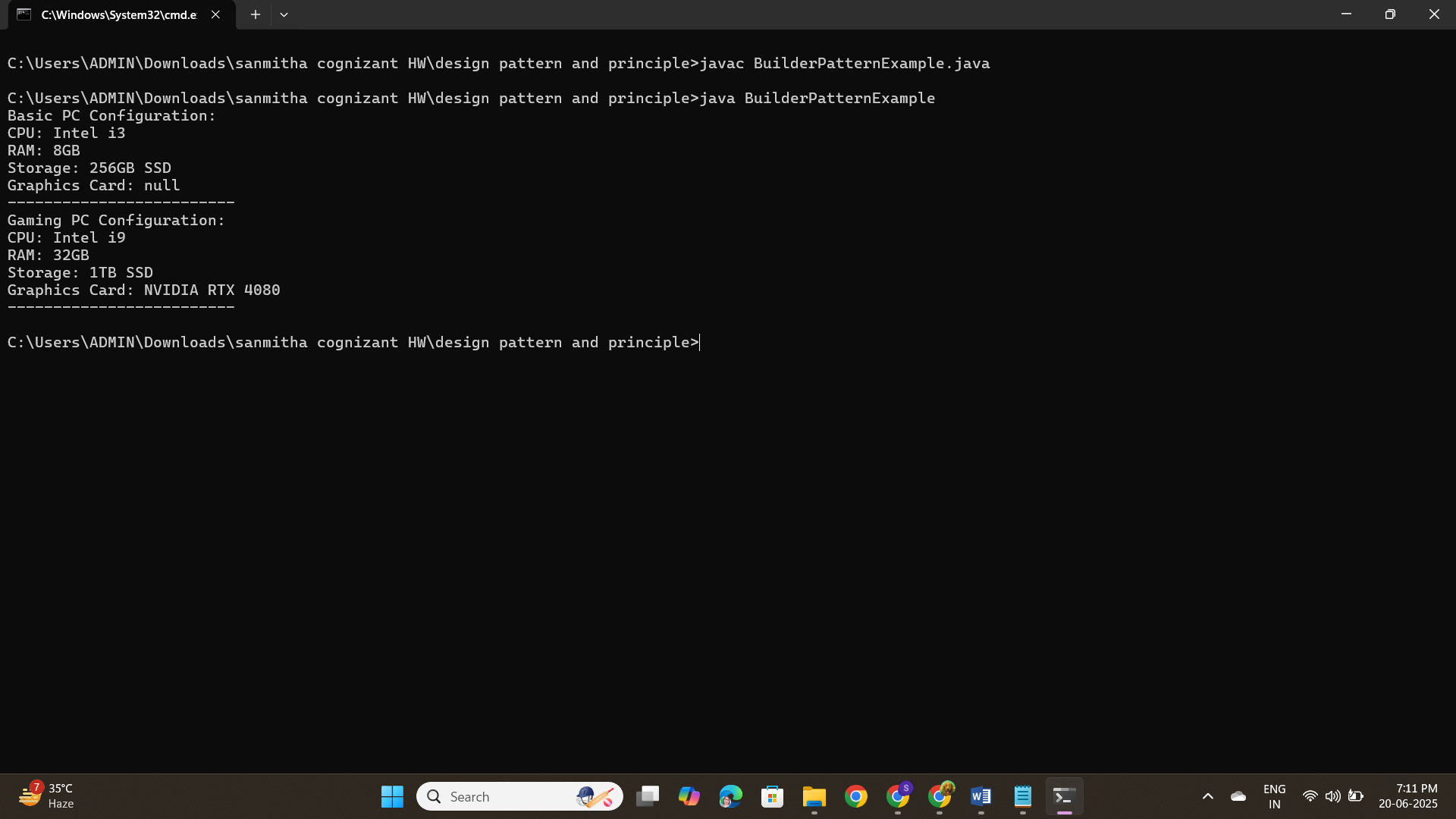
System.out.println("Gaming PC Configuration:");

gamingPC.showConfig();

}

}

**OUTPUT:**



**Exercise-4**

**CODE:**

public class AdapterPatternExample {

interface PaymentProcessor {

void processPayment(double amount);

}

static class PayPalGateway {

public void sendPayment(double amount) {

System.out.println("Processing payment via PayPal: ₹" + amount);

}

}

static class StripeGateway {

public void makeStripePayment(double amount) {

System.out.println("Processing payment via Stripe: ₹" + amount);

}

}

static class PayPalAdapter implements PaymentProcessor {

private PayPalGateway paypal;

public PayPalAdapter(PayPalGateway paypal) {

this.paypal = paypal;

}

public void processPayment(double amount) {

paypal.sendPayment(amount);

}

}

static class StripeAdapter implements PaymentProcessor {

private StripeGateway stripe;

public StripeAdapter(StripeGateway stripe) {

this.stripe = stripe;

}

public void processPayment(double amount) {

stripe.makeStripePayment(amount);

}

}

public static void main(String[] args) {

PaymentProcessor paypalProcessor = new PayPalAdapter(new PayPalGateway());

paypalProcessor.processPayment(500.0);

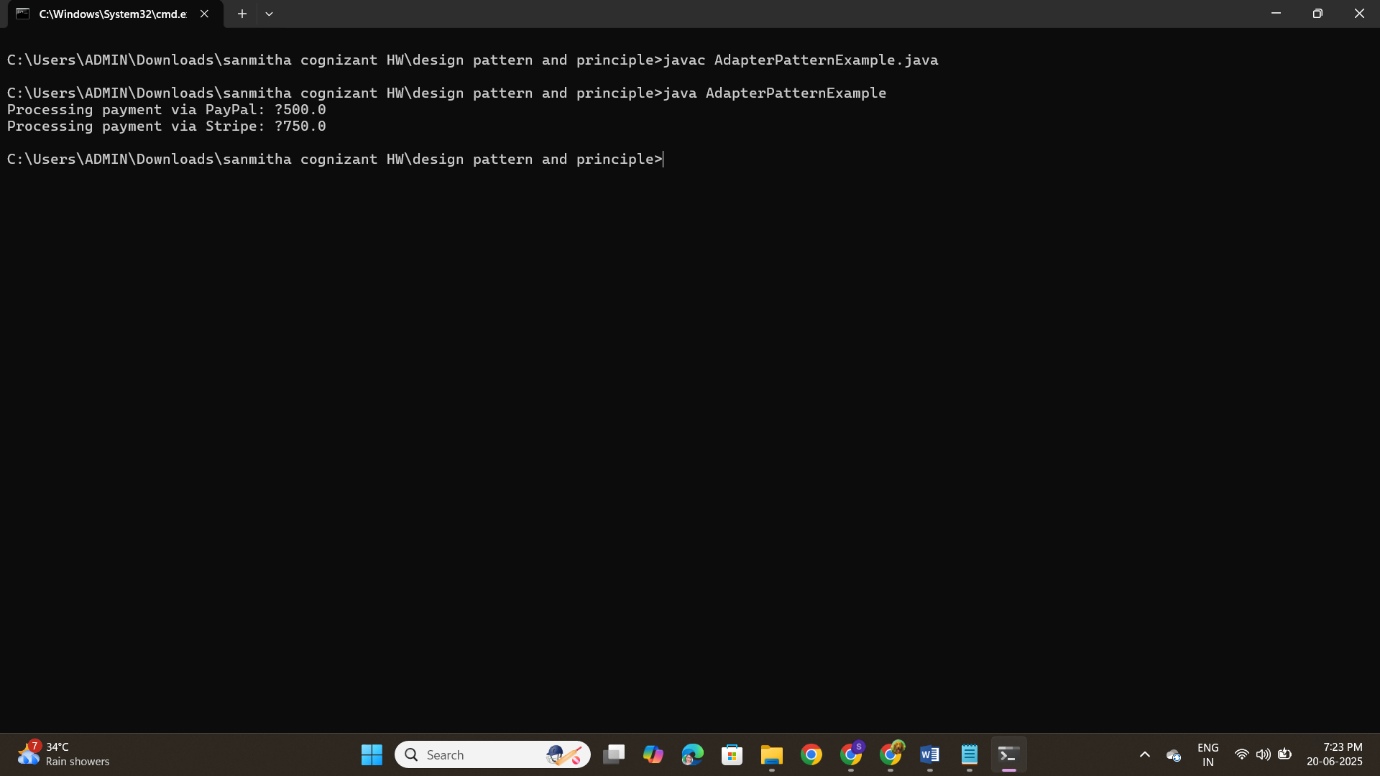
PaymentProcessor stripeProcessor = new StripeAdapter(new StripeGateway());

stripeProcessor.processPayment(750.0);

}

}

**OUTPUT:**

****

**Exercise-5**

**Code:**

public class DecoratorPatternExample {

interface Notifier {

void send(String message);

}

static class EmailNotifier implements Notifier {

public void send(String message) {

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

}

}

static abstract class NotifierDecorator implements Notifier {

protected Notifier notifier;

public NotifierDecorator(Notifier notifier) {

this.notifier = notifier;

}

public void send(String message) {

notifier.send(message);

}

}

static class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

static class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

public static void main(String[] args) {

Notifier notifier = new SlackNotifierDecorator(

new SMSNotifierDecorator(

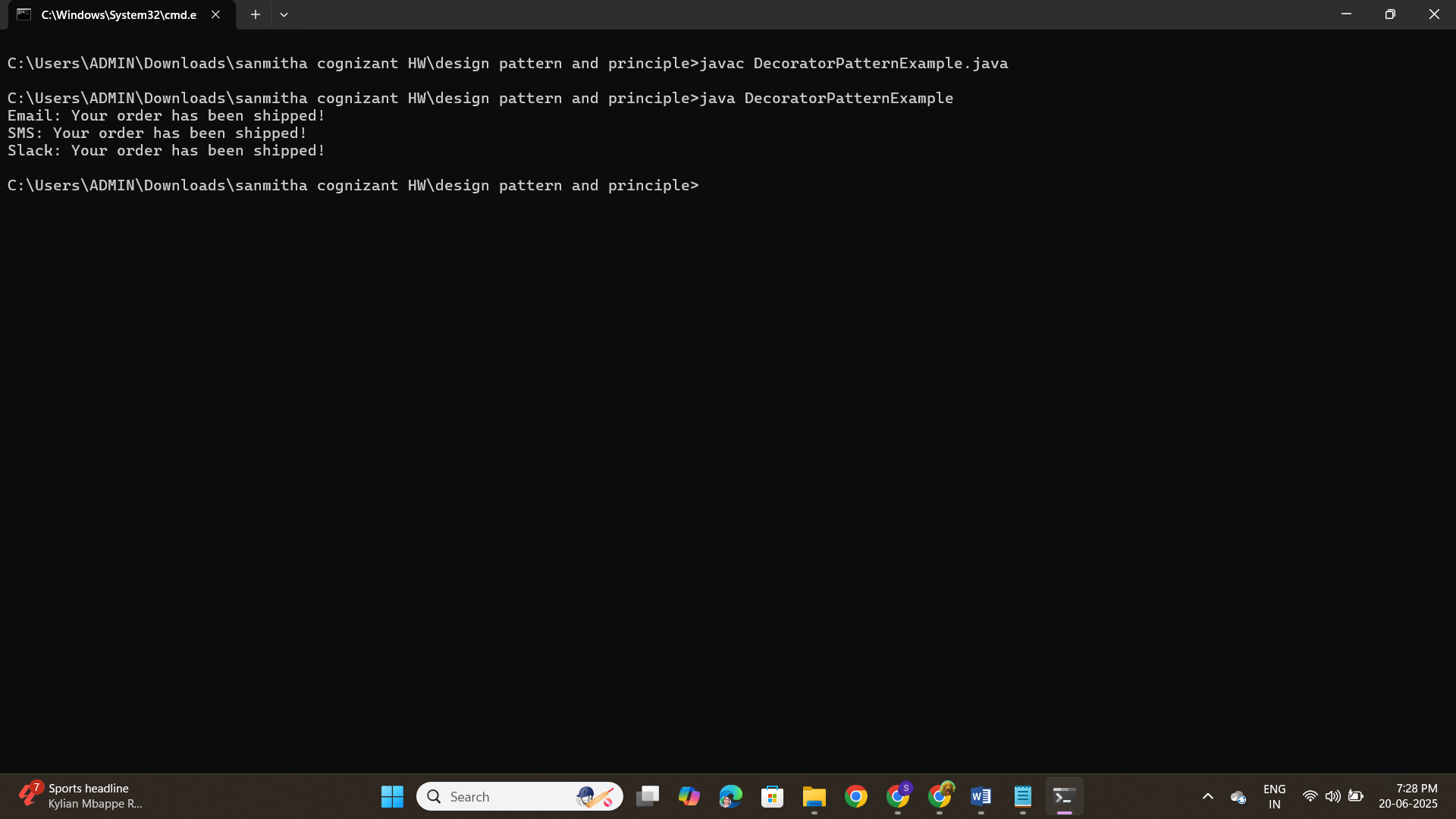
new EmailNotifier()));

notifier.send("Your order has been shipped!");

}

}

**Output:**

****

**Exercise-6**

**CODE:**

public class ProxyPatternExample {

interface Image {

void display();

}

static class RealImage implements Image {

private String filename;

public RealImage(String filename) {

this.filename = filename;

loadFromServer();

}

private void loadFromServer() {

System.out.println("Loading image from server: " + filename);

}

public void display() {

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

}

}

static class ProxyImage implements Image {

private RealImage realImage;

private String filename;

public ProxyImage(String filename) {

this.filename = filename;

}

public void display() {

if (realImage == null) {

realImage = new RealImage(filename);

}

realImage.display();

}

}

public static void main(String[] args) {

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

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

image1.display();

image1.display();

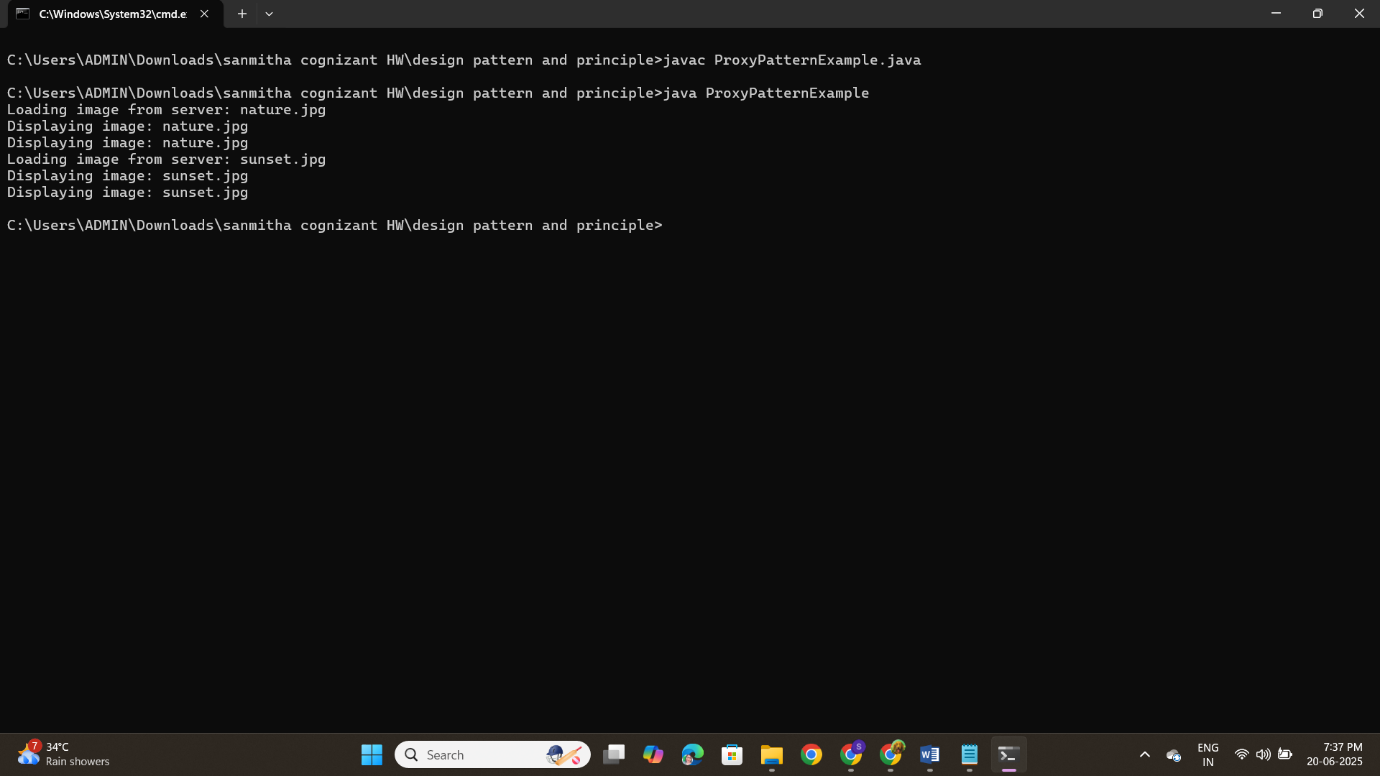
image2.display();

image2.display();

}

}

**Output:**



**Exercise-7**

**CODE:**

import java.util.ArrayList;

import java.util.List;

public class ObserverPatternExample {

interface Stock {

void register(Observer o);

void deregister(Observer o);

void notifyObservers();

void setPrice(double price);

}

static class StockMarket implements Stock {

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

private double stockPrice;

public void register(Observer o) {

observers.add(o);

}

public void deregister(Observer o) {

observers.remove(o);

}

public void notifyObservers() {

for (Observer o : observers) {

o.update(stockPrice);

}

}

public void setPrice(double price) {

this.stockPrice = price;

notifyObservers();

}

}

interface Observer {

void update(double price);

}

static class MobileApp implements Observer {

public void update(double price) {

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

}

}

static class WebApp implements Observer {

public void update(double price) {

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

}

}

public static void main(String[] args) {

StockMarket market = new StockMarket();

Observer mobile = new MobileApp();

Observer web = new WebApp();

market.register(mobile);

market.register(web);

market.setPrice(101.5);

market.setPrice(105.0);

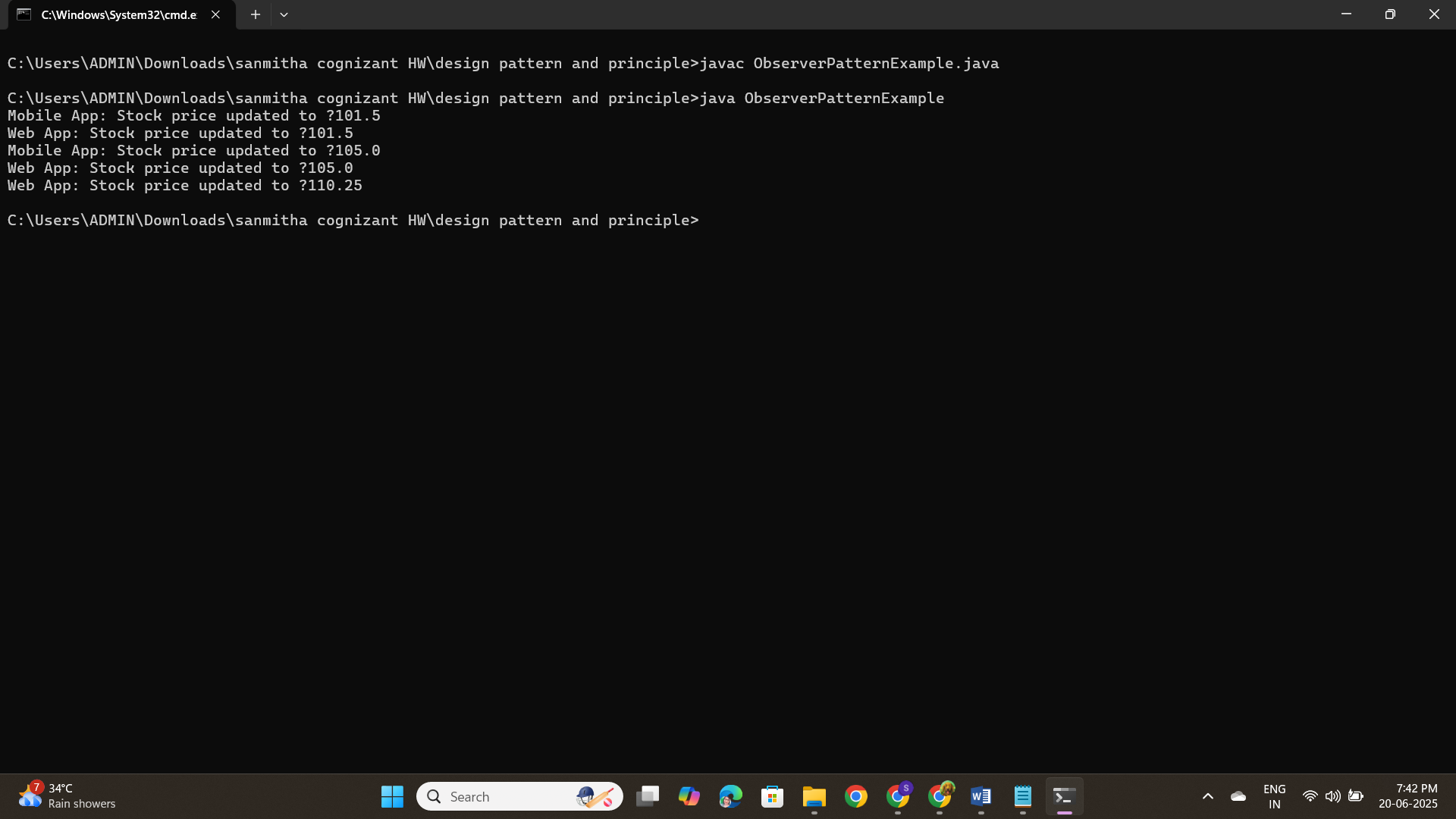
market.deregister(mobile);

market.setPrice(110.25);

}

}

**Output:**

****

**Exercise-8**

**Code:**

public class StrategyPatternExample {

interface PaymentStrategy {

void pay(double amount);

}

static class CreditCardPayment implements PaymentStrategy {

public void pay(double amount) {

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

}

}

static class PayPalPayment implements PaymentStrategy {

public void pay(double amount) {

System.out.println("Paid ₹" + amount + " using PayPal.");

}

}

static class PaymentContext {

private PaymentStrategy strategy;

public void setStrategy(PaymentStrategy strategy) {

this.strategy = strategy;

}

public void executePayment(double amount) {

strategy.pay(amount);

}

}

public static void main(String[] args) {

PaymentContext context = new PaymentContext();

context.setStrategy(new CreditCardPayment());

context.executePayment(500.0);

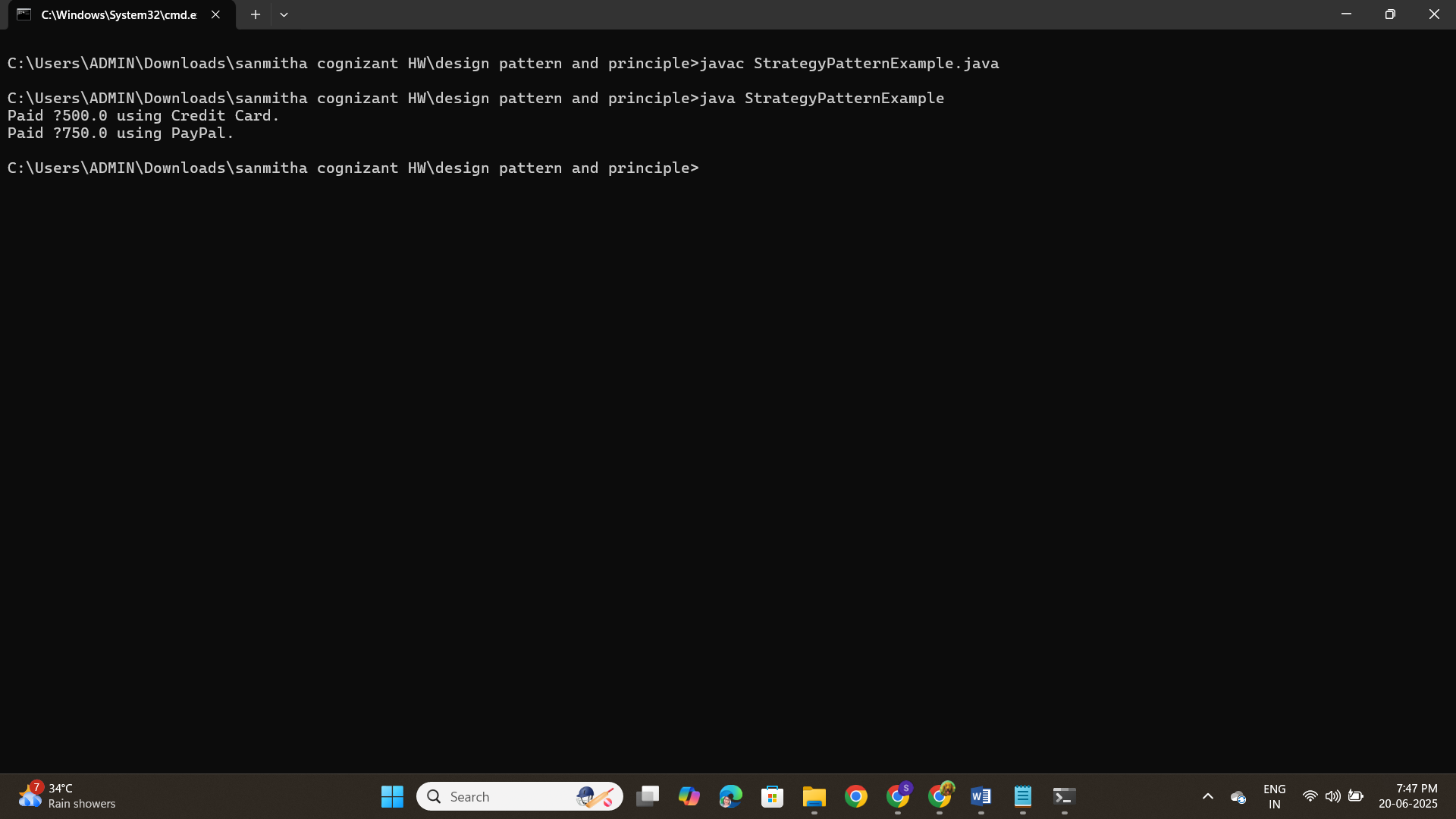
context.setStrategy(new PayPalPayment());

context.executePayment(750.0);

}

}

**Output:**

****

**Exercise-9**

**Code:**

public class CommandPatternExample {

interface Command {

void execute();

}

static class Light {

public void turnOn() {

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

}

public void turnOff() {

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

}

}

static class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOn();

}

}

static class LightOffCommand implements Command {

private Light light;

public LightOffCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOff();

}

}

static class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

command.execute();

}

}

public static void main(String[] args) {

Light light = new Light();

Command lightOn = new LightOnCommand(light);

Command lightOff = new LightOffCommand(light);

RemoteControl remote = new RemoteControl();

remote.setCommand(lightOn);

remote.pressButton();

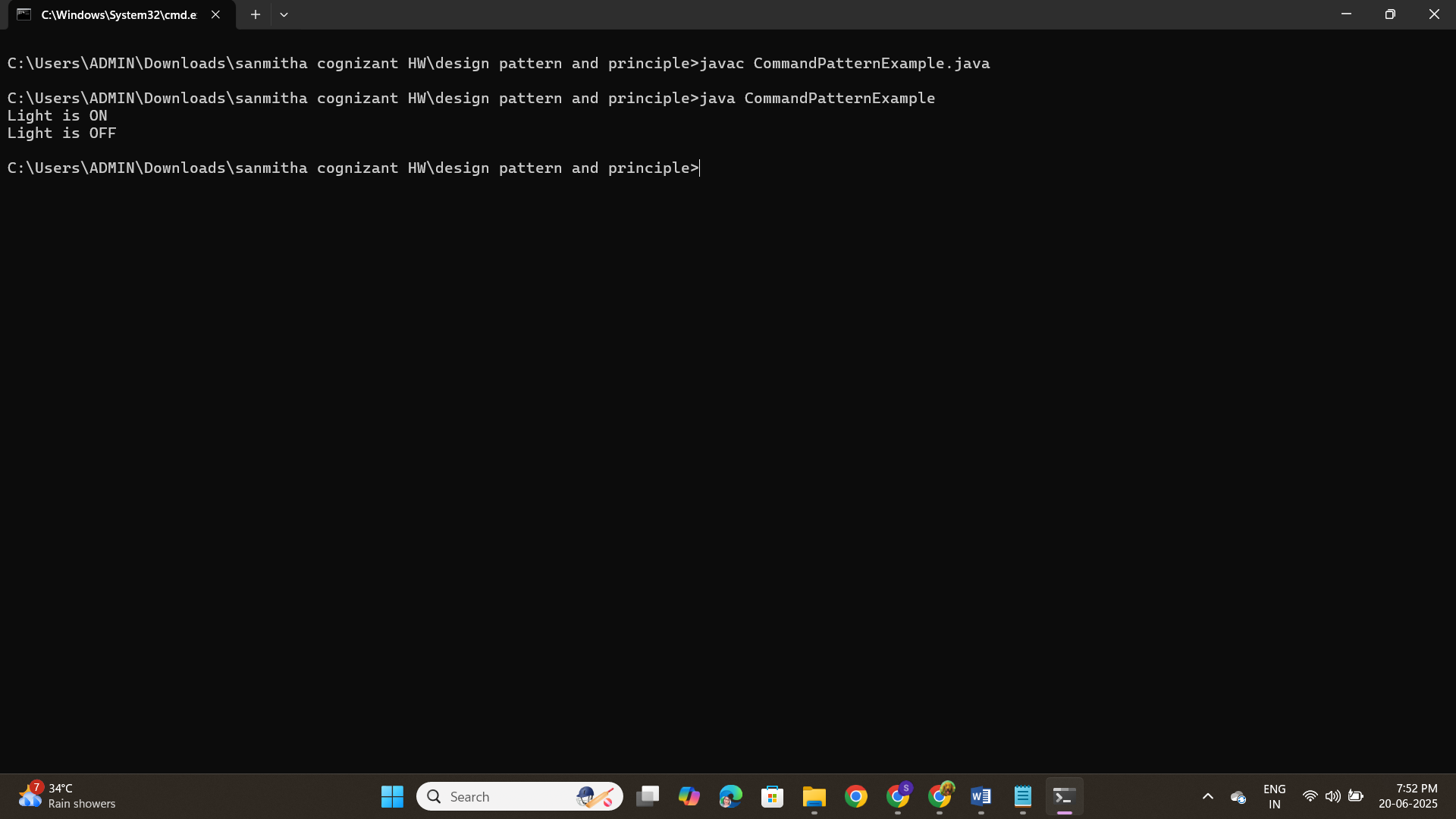
remote.setCommand(lightOff);

remote.pressButton();

}

}

**Output:**

****

**Exercise-10**

**Code:**

public class MVCPatternExample {

static class Student {

private String name;

private String id;

private String 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;

}

}

static 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);

}

}

static 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 setStudentId(String id) {

model.setId(id);

}

public void setStudentGrade(String grade) {

model.setGrade(grade);

}

public void updateView() {

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

}

}

public static void main(String[] args) {

Student student = new Student();

student.setName("Arjun");

student.setId("S123");

student.setGrade("A");

StudentView view = new StudentView();

StudentController controller = new StudentController(student, view);

controller.updateView();

controller.setStudentName("Meera");

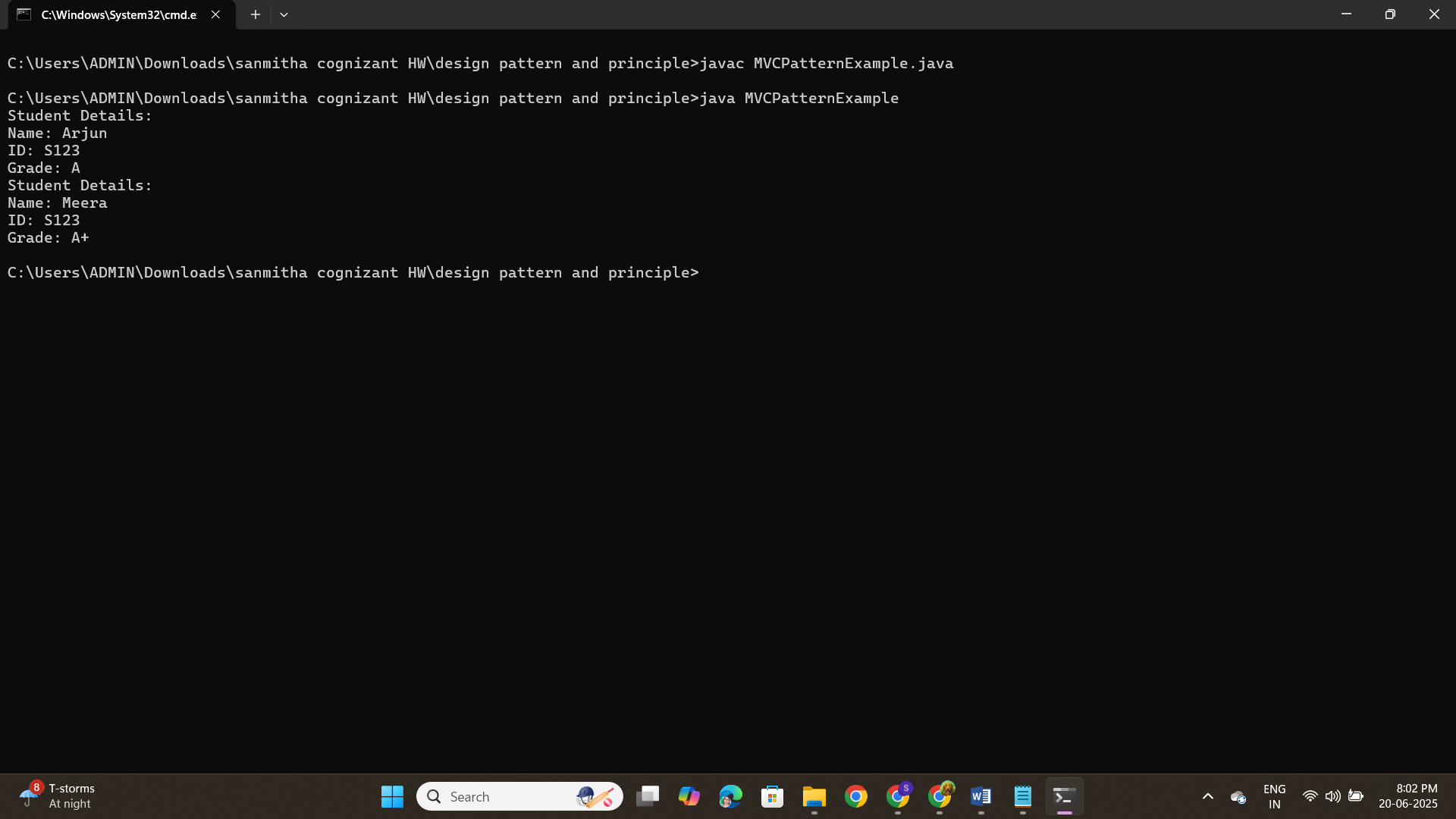
controller.setStudentGrade("A+");

controller.updateView();

}

}

**Output:**

****

**Exercise-11**

**Code:**

public class DependencyInjectionExample {

interface CustomerRepository {

String findCustomerById(String id);

}

static class CustomerRepositoryImpl implements CustomerRepository {

public String findCustomerById(String id) {

return "Customer[ID=" + id + ", Name=Ravi Kumar]";

}

}

static class CustomerService {

private CustomerRepository repository;

public CustomerService(CustomerRepository repository) {

this.repository = repository;

}

public void getCustomerDetails(String id) {

String customer = repository.findCustomerById(id);

System.out.println(customer);

}

}

public static void main(String[] args) {

CustomerRepository repo = new CustomerRepositoryImpl();

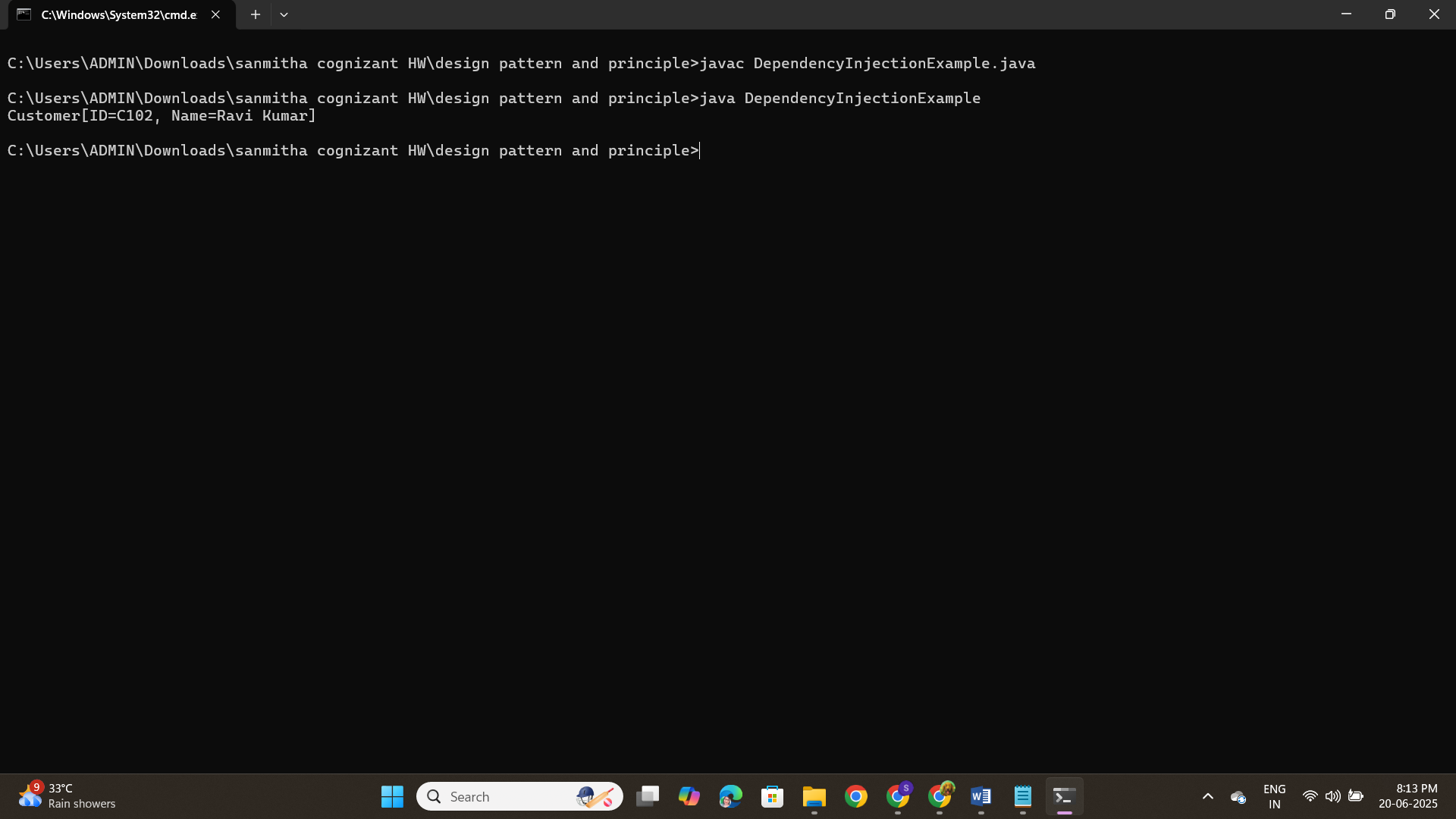
CustomerService service = new CustomerService(repo);

service.getCustomerDetails("C102");

}

}

**Output:**

****