**Exercise 1: Implementing the Singleton Pattern:**

**Logger.java:**

public class Logger {

private static Logger instance;

// Private constructor

public Logger() {

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

}

// Public method to get the instance

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void log(String message) {

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

}

}

**TestLogger.java:**

public class TestLogger {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("Message from logger1");

Logger logger2 = Logger.getInstance();

logger2.log("Message from logger2");

if (logger1 == logger2) {

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

}

else {

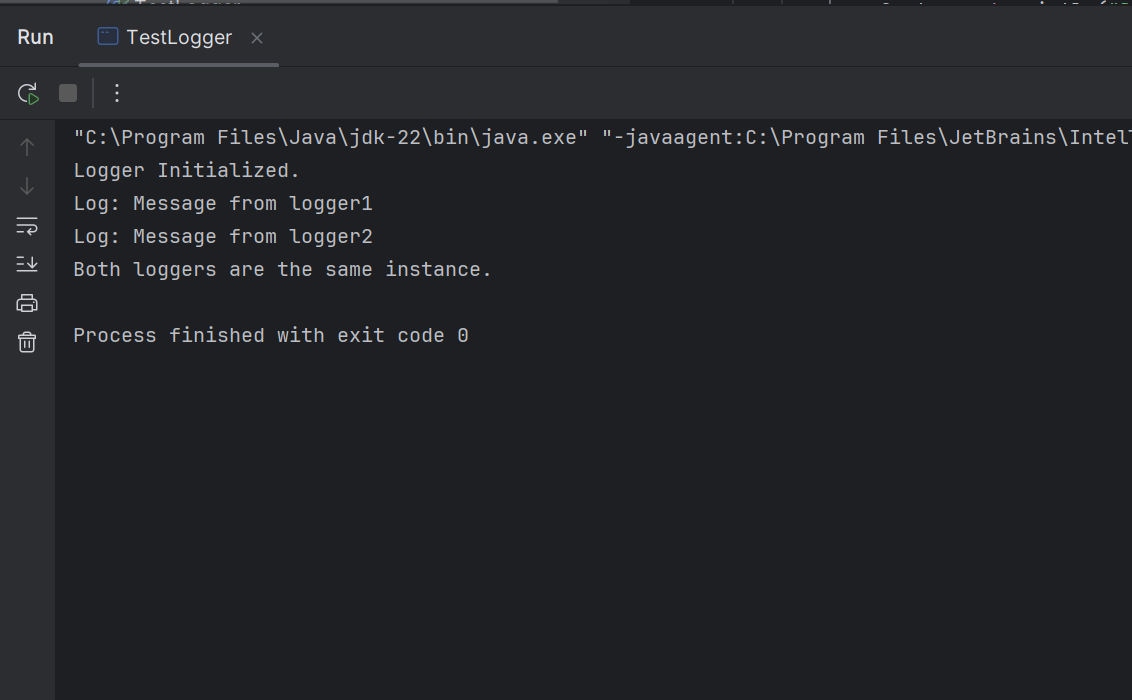
System.out.println("Different logger instances.");

}

}

}

**OUTPUT:**



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

**WordDocument.java:**

public class WordDocument implements Document {

@Override

public void open() {

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

}

}

**PdfDocument.java:**

public class PdfDocument implements Document {

@Override

public void open() {

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

}

}

**ExcelDocument.java:**

public class ExcelDocument implements Document {

@Override

public void open() {

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

}

}

**DocumentFactory.java:**

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordDocumentFactory.java:**

public class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java:**

public class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java:**

public class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

**FactoryTest.java:**

public class FactoryTest {

public static void main(String[] args) {

DocumentFactory wordF = new WordDocumentFactory();

Document word = wordF.createDocument();

word.open();

DocumentFactory pdfF = new PdfDocumentFactory();

Document pdf = pdfF.createDocument();

pdf.open();

DocumentFactory excelF = new ExcelDocumentFactory();

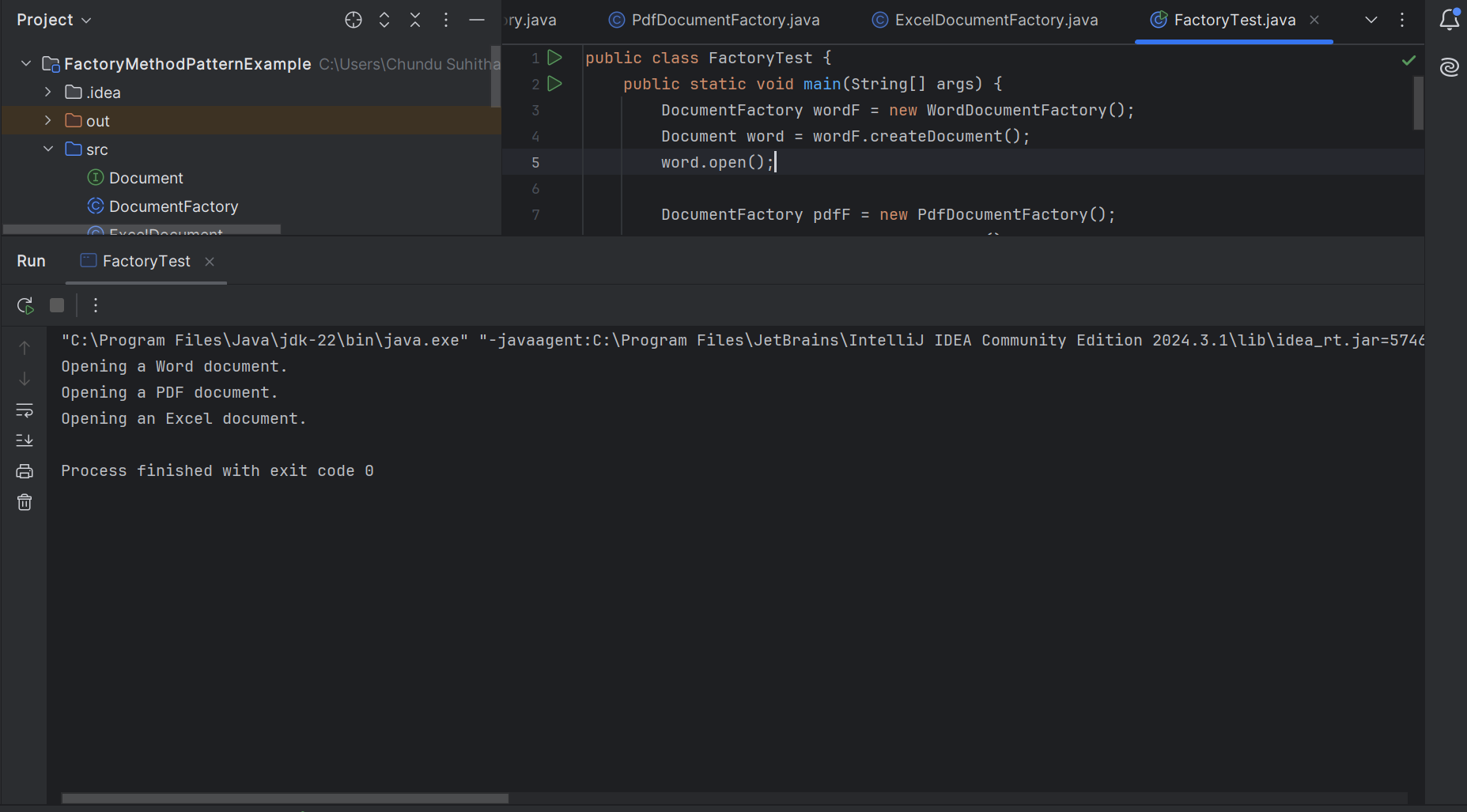
Document excel = excelF.createDocument();

excel.open();

}

}

**OUTPUT:**



**Exercise 3: Implementing the Builder Pattern**

**Computer.java:**

public class Computer {

private String CPU;

private String RAM;

private String storage;

private String graphicsCard;

private String keyboard;

private Computer(Builder builder) {

this.CPU = builder.CPU;

this.RAM = builder.RAM;

this.storage = builder.storage;

this.graphicsCard = builder.graphicsCard;

this.keyboard = builder.keyboard;

}

public static class Builder {

private String CPU;

private String RAM;

private String storage;

private String graphicsCard;

private String keyboard;

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 Builder setKeyboard(String keyboard) {

this.keyboard = keyboard;

return this;

}

public Computer build() {

return new Computer(this);

}

}

@Override

public String toString() {

return "Computer Configuration:\n" +

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

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

"Storage: " + (storage != null ? storage : "Not Included") + "\n" +

"Graphics Card: " + (graphicsCard != null ? graphicsCard : "Not Included") + "\n" +

"Keyboard: " + (keyboard != null ? keyboard : "Not Included");

}

}

**TestBulider.java:**

public class TestBuilder {

public static void main(String[] args) {

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

System.out.println(basicComputer);

System.out.println();

Computer gamingComputer = new Computer.Builder("Intel i9", "32GB")

.setStorage("1TB SSD")

.setGraphicsCard("NVIDIA RTX 4090")

.setKeyboard("Mechanical RGB")

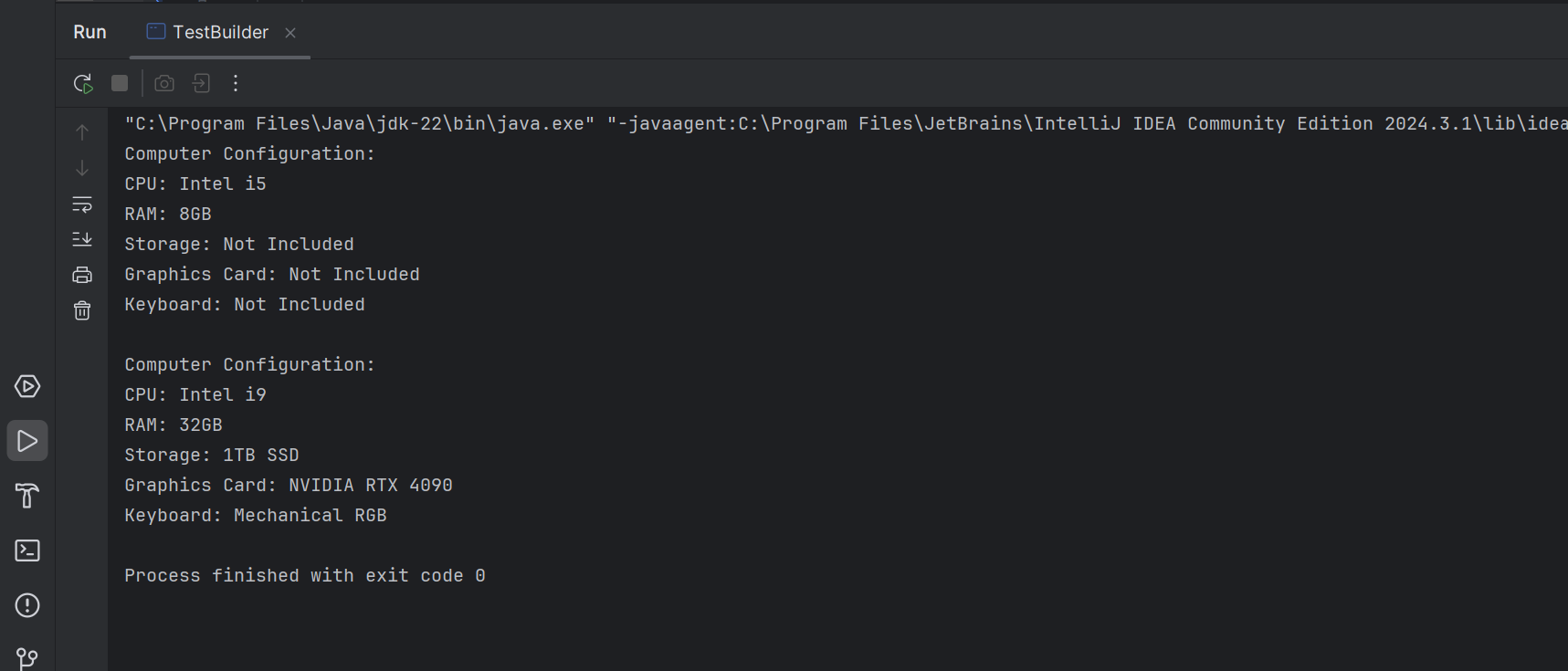
.build();

System.out.println(gamingComputer);

}

}

**OUTPUT:**



**Exercise 4: Implementing the Adapter Pattern:**

**PayPalGateway.java:**

public class PayPalGateway {

public void makePayment(double amount) {

System.out.println("Payment of ₹" + amount + " processed via PayPal.");

}

}

**PhonepeGateway.java:**

public class PhonePeGateway {

public void doTransaction(double amount) {

System.out.println("Payment of ₹" + amount + " processed via PhonePe.");

}

}

**PayPalAdapter.java:**

public class PayPalAdapter implements PaymentProcessor {

private PayPalGateway paypal;

public PayPalAdapter(PayPalGateway paypal) {

this.paypal = paypal;

}

@Override

public void processPayment(double amount) {

paypal.makePayment(amount);

}

}

**PhonePeAdapter.java:**

public class PhonePeAdapter implements PaymentProcessor {

private PhonePeGateway phonePe;

public PhonePeAdapter(PhonePeGateway phonePe) {

this.phonePe = phonePe;}

@Override

public void processPayment(double amount) {

phonePe.doTransaction(amount);

}

}

**AdapterTest.java:**

public class AdapterTest {

public static void main(String[] args) {

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

PaymentProcessor phonePe = new PhonePeAdapter(new PhonePeGateway());

System.out.println("Using PayPal Adapter:");

paypl.processPayment(1000.0);

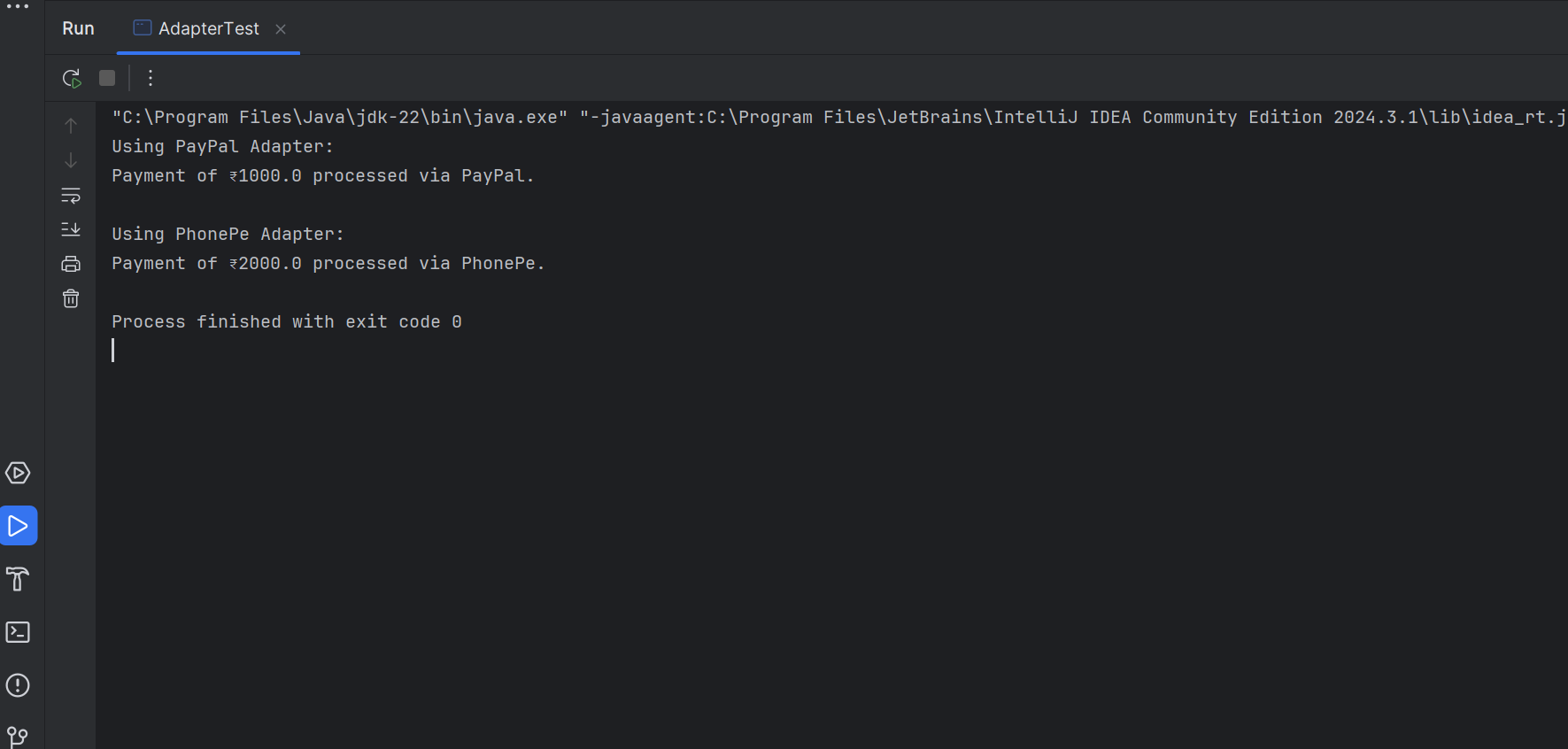
System.out.println("\nUsing PhonePe Adapter:");

phonePe.processPayment(2000.0);

}

}

**OUTPUT:**



**Exercise 5: Implementing the Decorator Pattern**

**Main.java:**

public class Main {

public static void main(String[] args) {

Notifier baseNotifier = new EmailNotifier();

// Add SMS notification

Notifier smsNotifier = new SMSNotifierDecorator(baseNotifier);

// Add Slack notification on top of SMS and Email

Notifier slackNotifier = new SlackNotifierDecorator(smsNotifier);

slackNotifier.send("System Alert: Low Disk Space!");

}

}

**EmailNotifier.java:**

public class EmailNotifier implements Notifier {

@Override

public void send(String message) {

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

}

}

**NotifierDecorator.java:**

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);

}

}

**SlackNotifierDecorator.java:**

public class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

private void sendSlack(String message) {

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

}

@Override

public void send(String message) {

super.send(message); // call wrapped notifier

sendSlack(message); // additional behavior

}

}

**SMSNotifierDecorator.java:**

public class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

private void sendSMS(String message) {

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

}

@Override

public void send(String message) {

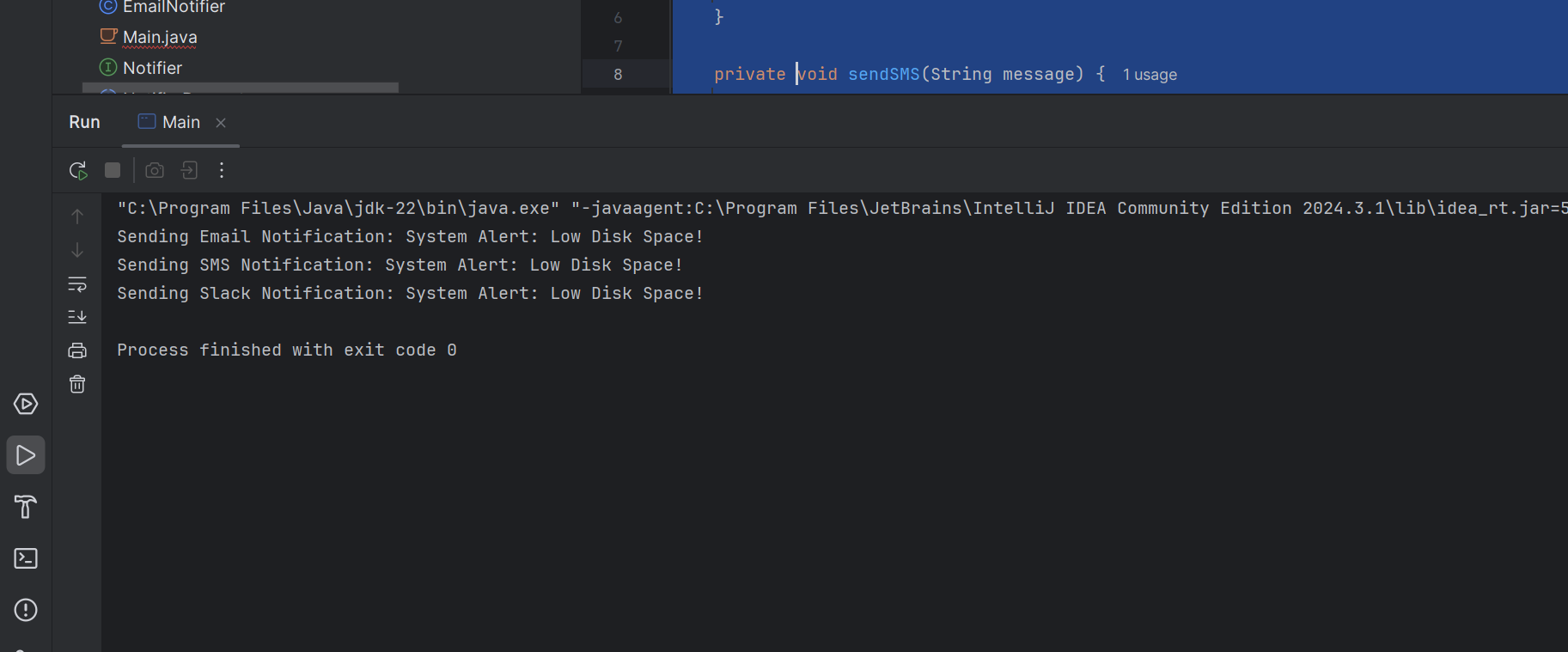
super.send(message); // call wrapped notifier

sendSMS(message); // additional behavior

}

}

**OUTPUT:**



**Exercise 6: Implementing the Proxy Pattern**

**RealImage.java:**

public 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);

}

@Override

public void display() {

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

}

}

**ProxyImage.java:**

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("Image loaded from cache: " + filename);

}

realImage.display();

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

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

image1.display();

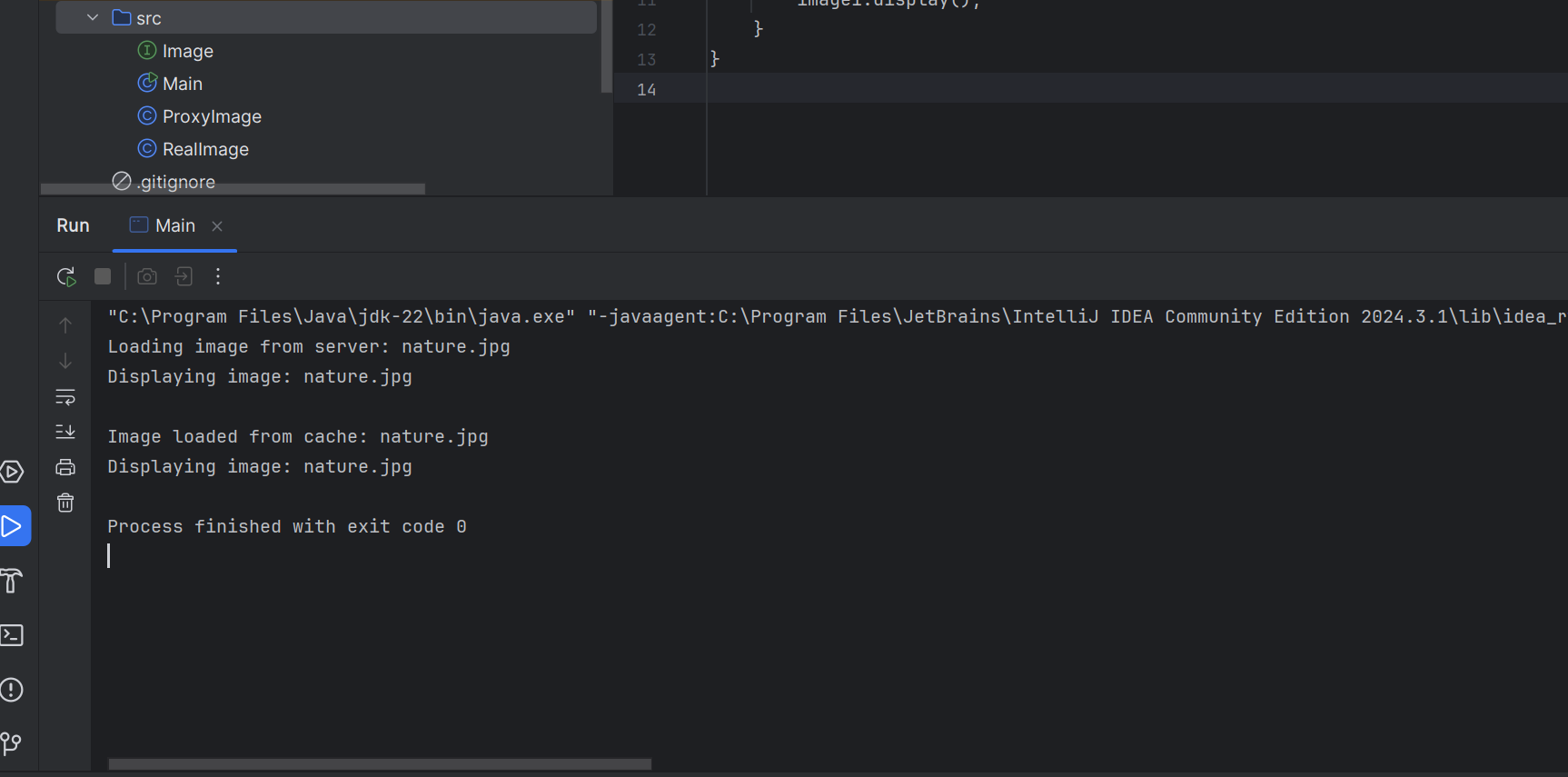
System.out.println();

image1.display();

}

}

**OUTPUT:**



**Exercise 7: Implementing the Observer Pattern**

**MobileApp.java:**

public class MobileApp implements Observer {

private String appName;

public MobileApp(String appName) {

this.appName = appName;

}

@Override

public void update(String stockName, double stockPrice) {

System.out.println(appName + " - Mobile Notification: " + stockName + " price changed to ₹" + stockPrice);

}

}

**StockMarket.java:**

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;

public void setStockData(String stockName, double stockPrice) {

this.stockName = stockName;

this.stockPrice = stockPrice;

notifyObservers();

}

public String getStockName() {

return stockName;

}

public double getStockPrice() {

return stockPrice;

}

@Override

public void registerObserver(Observer o) {

observers.add(o);

}

@Override

public void deregisterObserver(Observer o) {

observers.remove(o);

}

@Override

public void notifyObservers() {

for (Observer o : observers) {

o.update(stockName, stockPrice);

}

}

}

**WebApp.java:**

public class WebApp implements Observer {

private String appName;

public WebApp(String appName) {

this.appName = appName;

}

@Override

public void update(String stockName, double stockPrice) {

System.out.println(appName + " - Web Notification: " + stockName + " price updated to ₹" + stockPrice);

}

}

**Stock.java:**

public interface Stock {

void registerObserver(Observer o);

void deregisterObserver(Observer o);

void notifyObservers();

}

**Main.java:**

public class Main {

public static void main(String[] args) {

StockMarket stockMarket = new StockMarket();

Observer mobileApp = new MobileApp("StocksNow");

Observer webApp = new WebApp("MarketWatch");

stockMarket.registerObserver(mobileApp);

stockMarket.registerObserver(webApp);

stockMarket.setStockData("Digital", 3720.10);

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

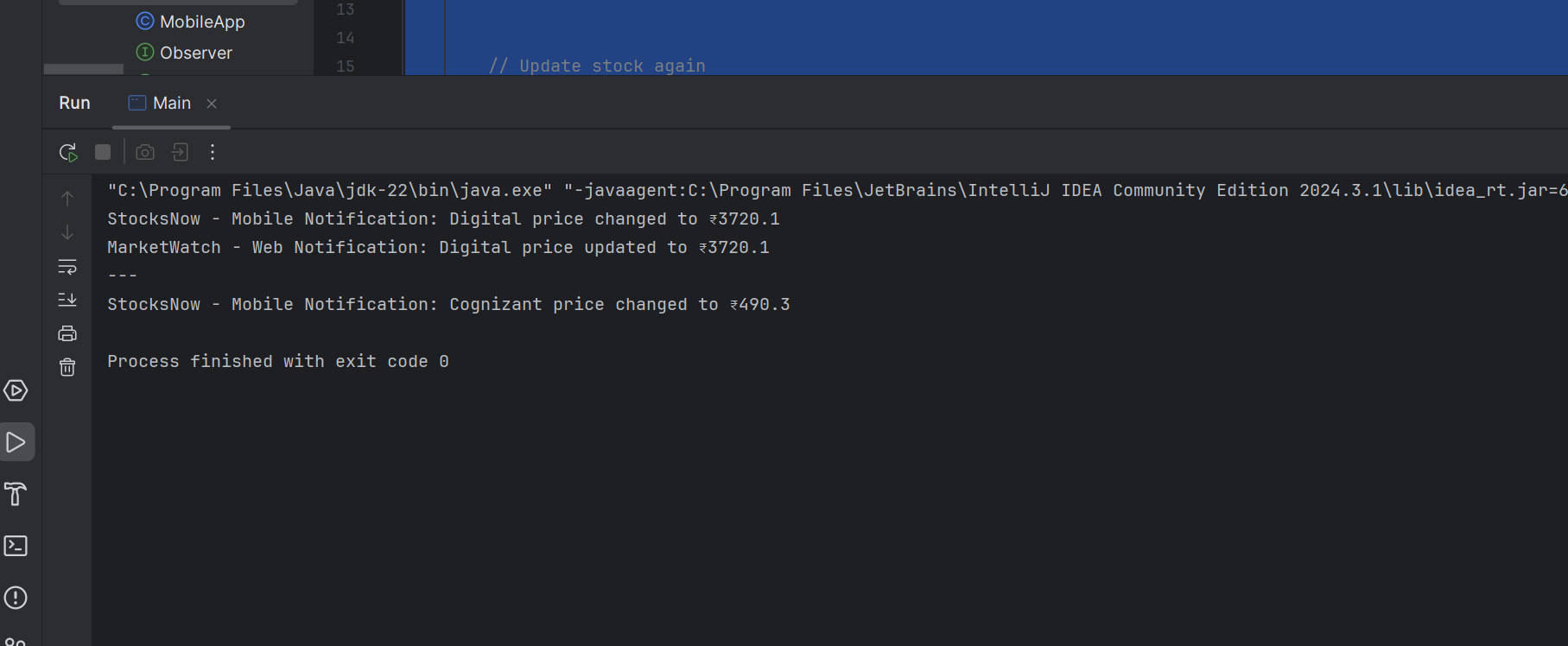
stockMarket.deregisterObserver(webApp);

stockMarket.setStockData("Cognizant", 490.30);

}

}

**OUTPUT:**



**Exercise 8: Implementing the Strategy Pattern**

**PayPalPayment.java:**

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:**

public class PaymentContext {

private PaymentStrategy strategy;

public void setPaymentStrategy(PaymentStrategy strategy) {

this.strategy = strategy;

}

public void processPayment(double amount) {

if (strategy == null) {

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

} else {

strategy.pay(amount);

}

}

}

**CreditCardPayment.java:**

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 [" + cardNumber + "] - " + cardHolder);

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

PaymentContext context = new PaymentContext();

PaymentStrategy creditCard = new CreditCardPayment("1234-5678-9012-3456", "suhitha chundu");

context.setPaymentStrategy(creditCard);

context.processPayment(2500.75);

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

PaymentStrategy payPal = new PayPalPayment("suhitga@example.com");

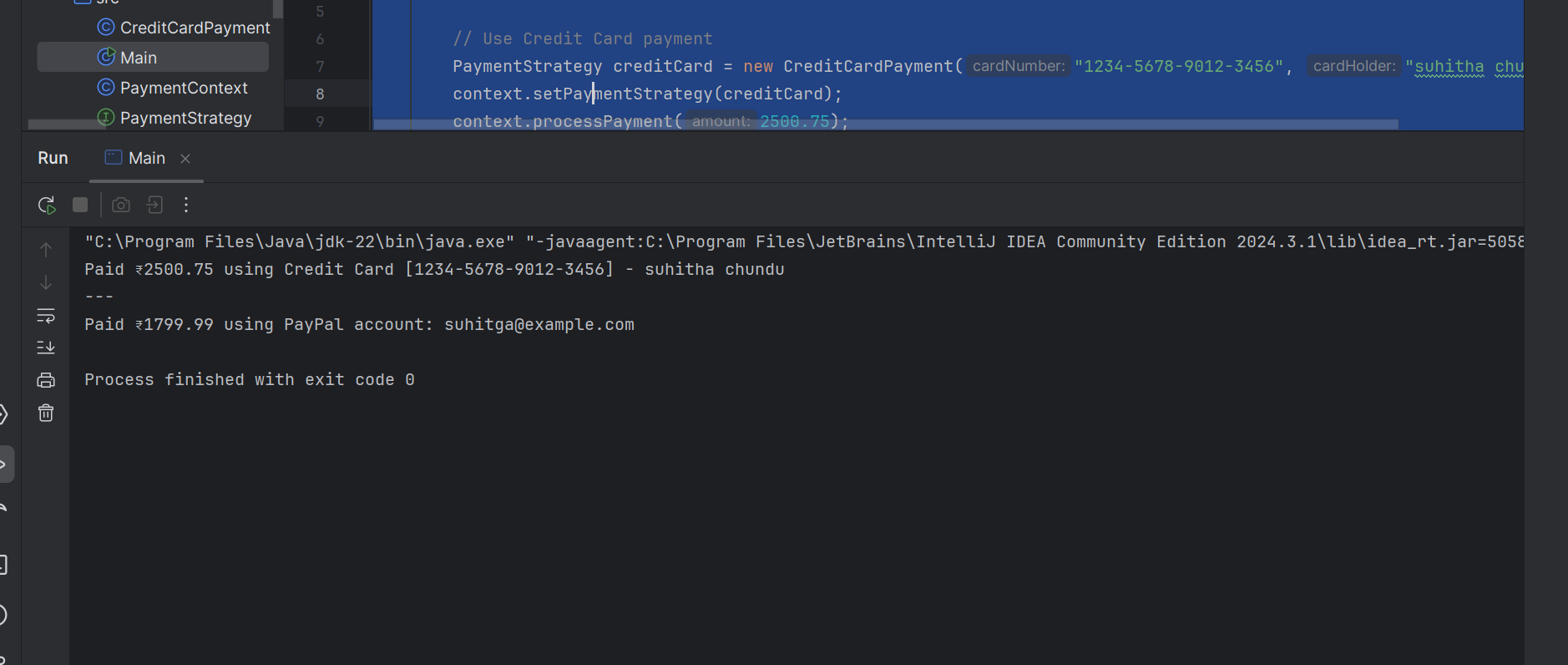
context.setPaymentStrategy(payPal);

context.processPayment(1799.99);

}

}

**OUTPUT:**



**Exercise 9: Implementing the Command Pattern**

**LightOnCommand.java:**

public class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

@Override

public void execute() {

light.turnOn();

}

}

**LightOffCommand.java:**

public class LightOffCommand implements Command {

private Light light;

public LightOffCommand(Light light) {

this.light = light;

}

@Override

public void execute() {

light.turnOff();

}

}

**RemoteControl.java:**

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 assigned.");

}

}

}

**Light.java:**

public class Light {

public void turnOn() {

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

}

public void turnOff() {

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

}

}

**Main.java:**

public class Main {

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();

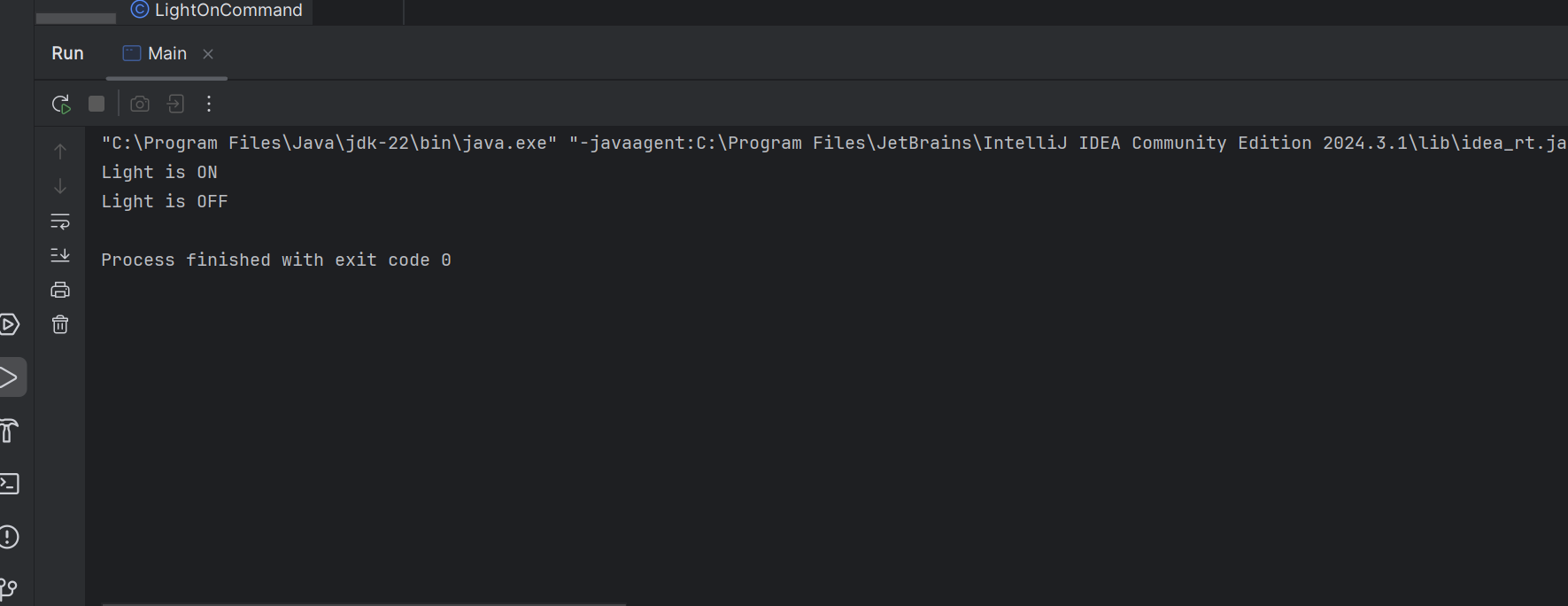
remote.setCommand(lightOff);

remote.pressButton();

}

}

**OUTPUT:**



**Exercise 10: Implementing the MVC Pattern**

**Student.java:**

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;

}

// Getters and Setters

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;

}

}

**StudentView.java:**

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();

}}

**StudentController.java:**

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());

}}

**Main.java:**

public class Main {

public static void main(String[] args) {

Student student = new Student("suhitha", "30487", "A");

StudentView view = new StudentView();

StudentController controller = new StudentController(student, view); controller.updateView();

controller.setStudentName("Suhitha Chundu");

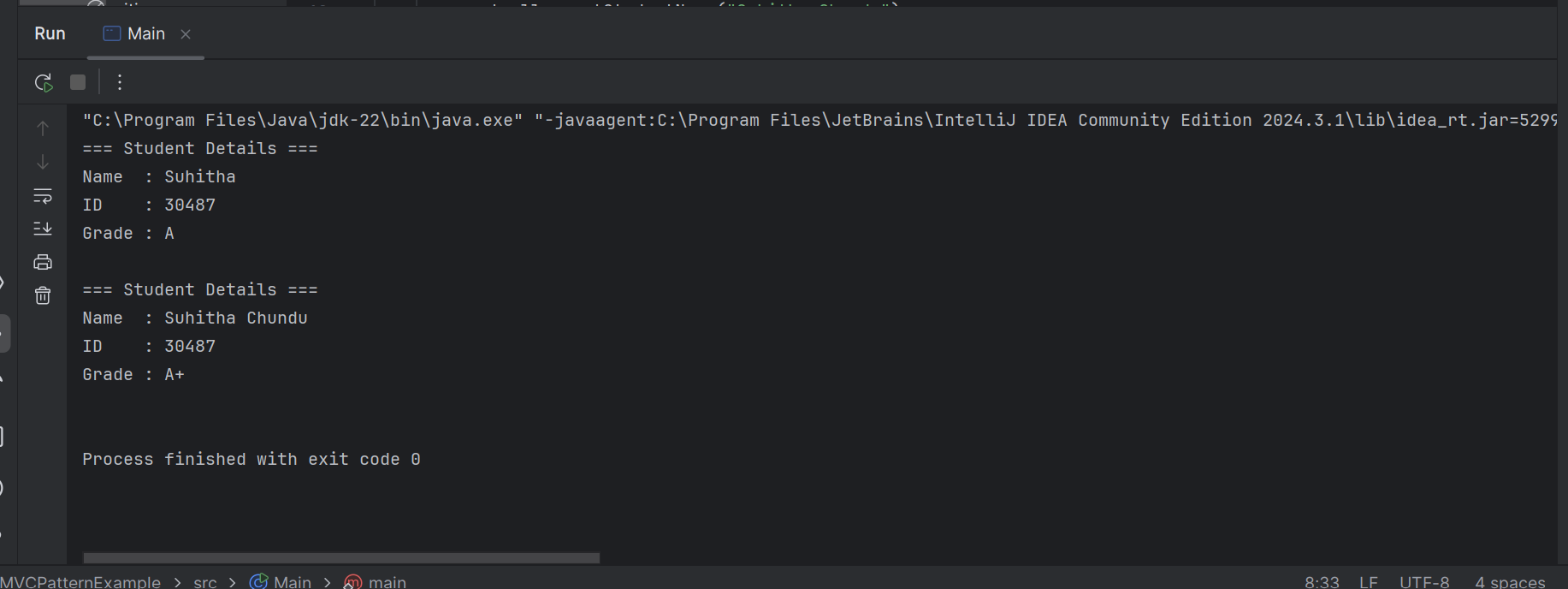
controller.setStudentGrade("A+");

controller.updateView();

}

}

**OUTPUT:**



**Exercise 11: Implementing Dependency Injection:**

**CustomerRepository.java:**

public class CustomerRepositoryImpl implements CustomerRepository {

@Override

public String findCustomerById(String customerId) {

// Simulated customer data

return "Customer [ID=" + customerId + ", Name=suhitha]";

}

}

**CustomerService.java:**

public class CustomerService {

private CustomerRepository customerRepository;

public CustomerService(CustomerRepository customerRepository) {

this.customerRepository = customerRepository; }

public void getCustomerDetails(String customerId) {

String customer = customerRepository.findCustomerById(customerId);

System.out.println("Retrieved: " + customer);

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

CustomerRepository repository = new CustomerRepositoryImpl();

CustomerService service = new CustomerService(repository);

service.getCustomerDetails("C102");

}}

**OUTPUT:**

