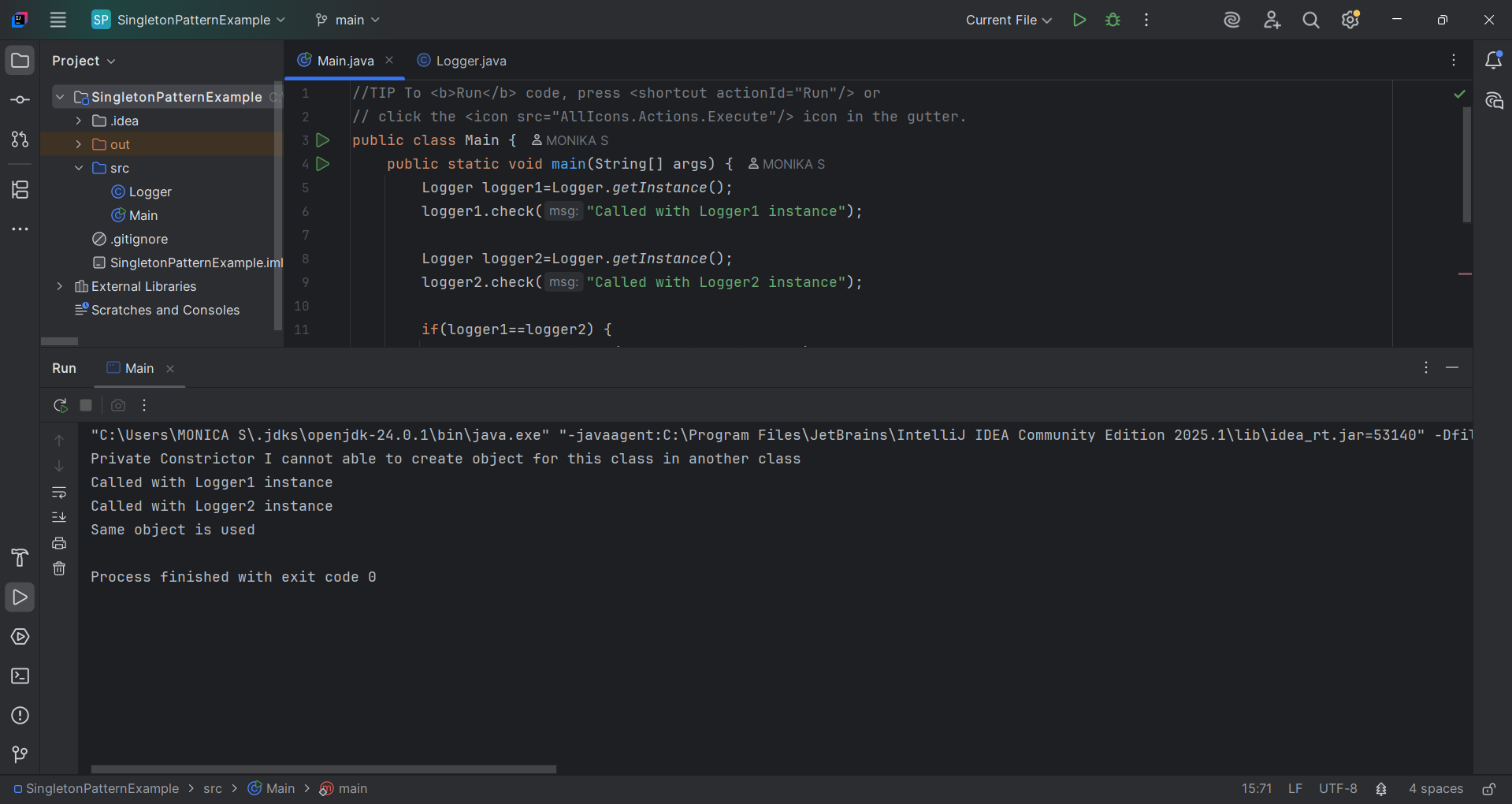
**1. SingletonPatternExample**

**Main.java**  
public class Main {  
 public static void main(String[] args) {  
 Logger logger1=Logger.*getInstance*();  
 logger1.check("Called with Logger1 instance");  
 Logger logger2=Logger.*getInstance*();  
 logger2.check("Called with Logger2 instance");  
 if(logger1==logger2) {  
 System.*out*.println("Same object is used");  
 }  
 else{  
 System.*out*.println("Not a singleton object pattern program");  
 }  
 }  
}

**Logger.java**

public class Logger {  
 private static Logger *log*;  
 private Logger() {  
 System.*out*.println("Private Constrictor I cannot able to create object for this class in another class");  
 }  
 public static Logger getInstance() {  
 if(*log*==null) {  
 *log*=new Logger();  
 return *log*;  
 }  
 return *log*;  
 }  
  
 public void check(String msg) {  
 System.*out*.println(msg);  
 }  
}



**2. FactoryMethodPatternExample**

**Main.java**  
public class Main {  
 public static void main(String[] args) {  
 DocumentFactory word=new WordDocumentFactory();  
 Document wordDoc=word.createDocument();  
 wordDoc.open();  
 wordDoc.close();  
  
 DocumentFactory pdf =new PdfDocumentfactory();  
 Document pdfDoc=pdf.createDocument();  
 pdfDoc.open();  
 pdfDoc.close();  
  
 DocumentFactory excel=new ExcelDocumentFactory();  
 Document excelDoc=excel.createDocument();  
 excelDoc.open();  
 excelDoc.close();  
 }  
}

**Document.java**

public interface Document {  
 void open();  
 void close();  
}

**DocumentFactory.java**

public abstract class DocumentFactory {  
 public abstract Document createDocument();  
}

**ExcelDocument.java**

public class ExcelDocument implements Document{  
 @Override  
 public void open() {  
 System.*out*.println("Opening Excel Document");  
 }  
  
 @Override  
 public void close() {  
 System.*out*.println("Closing Excel Document");  
 }  
}

**ExcelDocumentFactory.java**

public class ExcelDocumentFactory extends DocumentFactory{  
 @Override  
 public Document createDocument() {  
 return new ExcelDocument();  
 }  
}

**PdfDocument.java**

public class PdfDocument implements Document{  
 @Override  
 public void open() {  
 System.*out*.println("Opening PDF Document");  
 }  
  
 @Override  
 public void close() {  
 System.*out*.println("Closing PDF Document");  
 }  
}

**PdfDocumentfactory.java**

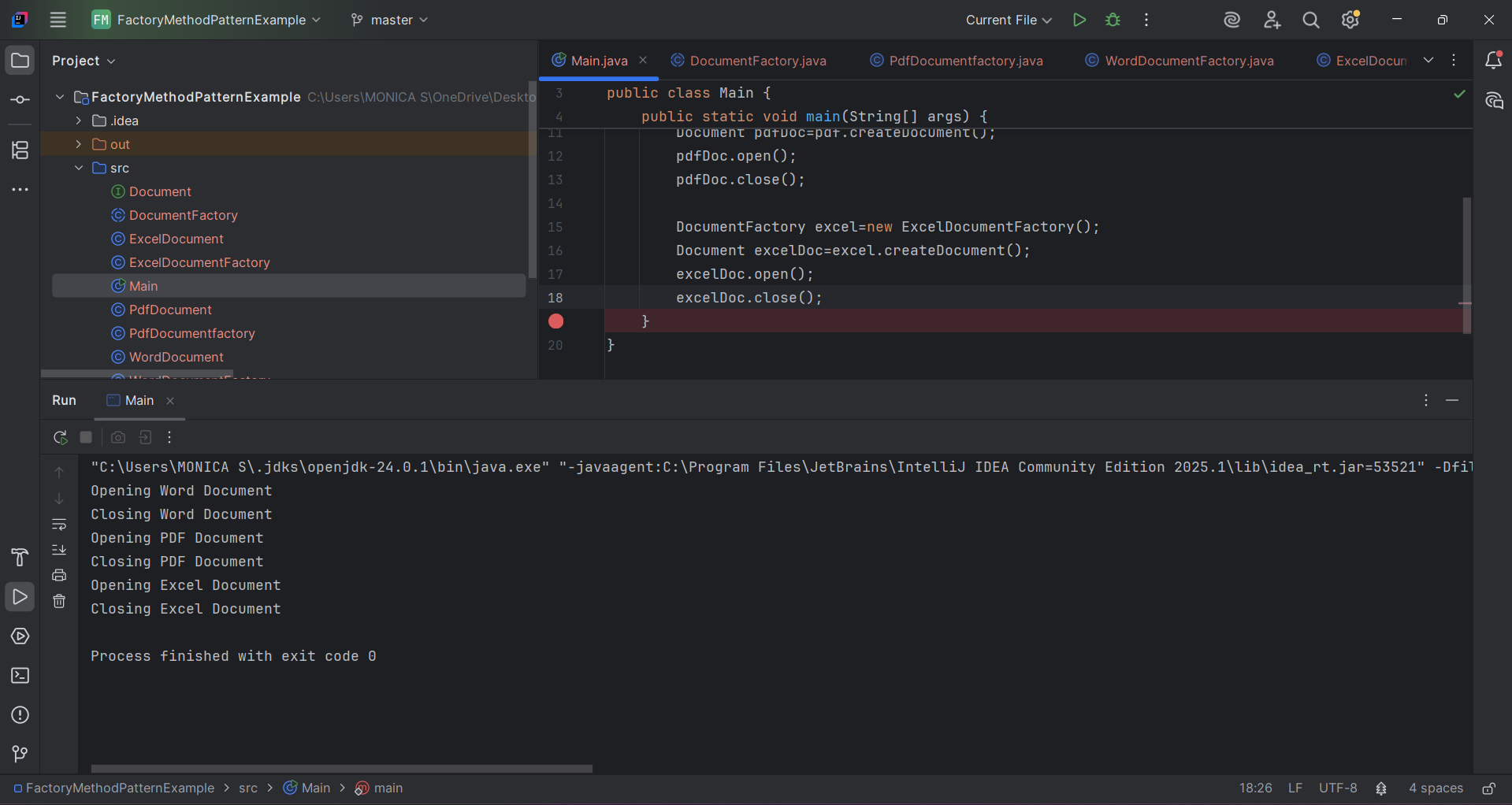
public class PdfDocumentfactory extends DocumentFactory{  
 @Override  
 public Document createDocument() {  
 return new PdfDocument();  
 }  
}

**WordDocumet.java**

public class WordDocument implements Document{  
 @Override  
 public void open() {  
 System.*out*.println("Opening Word Document");  
 }  
  
 @Override  
 public void close() {  
 System.*out*.println("Closing Word Document");  
 }  
}

**WordDocumentFactory.java**

public class WordDocumentFactory extends DocumentFactory{  
 @Override  
 public Document createDocument() {  
 return new WordDocument();  
 }  
}

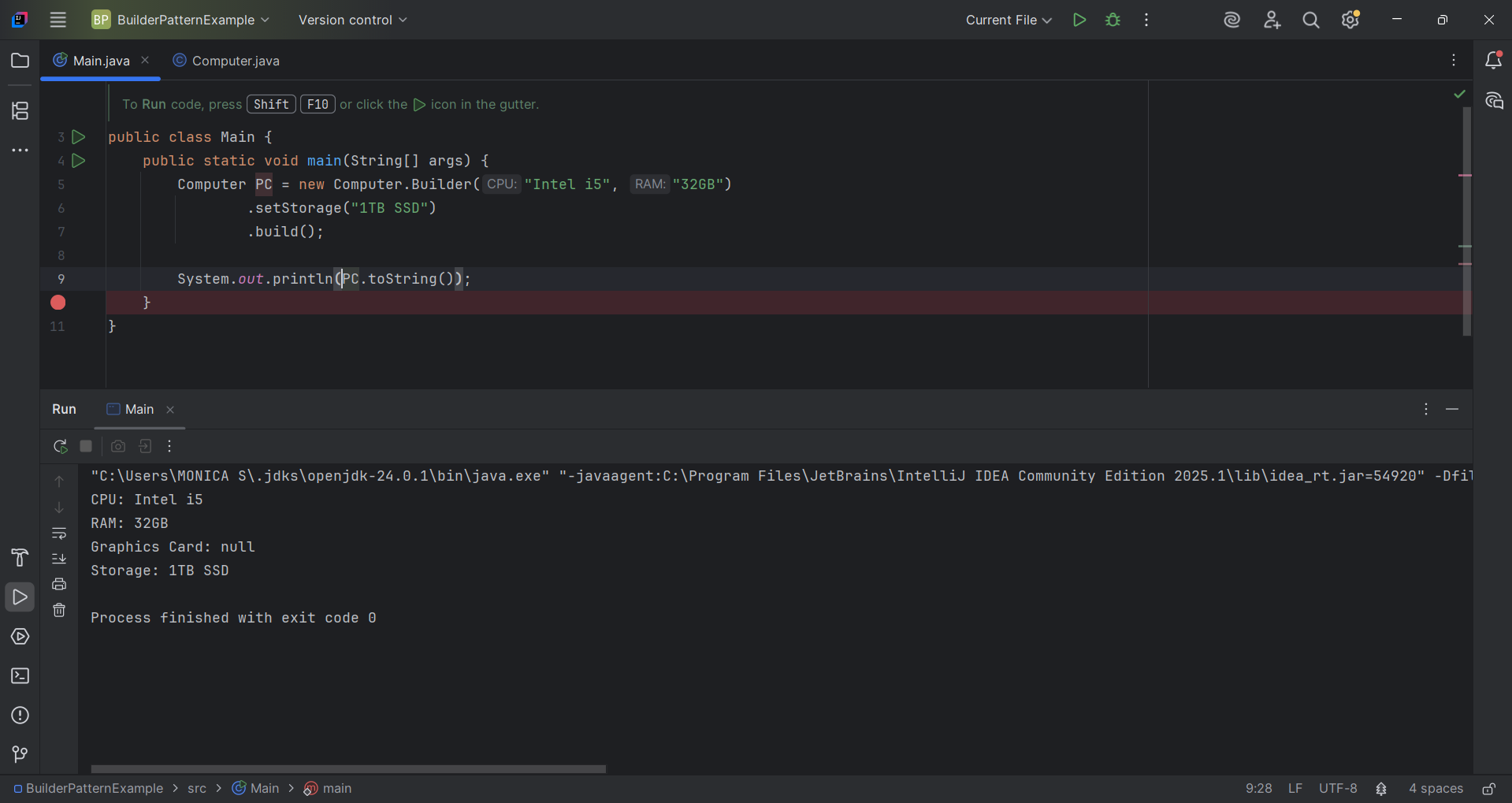


**3. BuilderPatternExample**

**Main.java**  
public class Main {  
 public static void main(String[] args) {  
 Computer PC = new Computer.Builder("Intel i5", "32GB")  
 .setStorage("1TB SSD")  
 .build();  
  
 System.out.println(PC.toString());  
 }  
}

**Computer.java**

import javax.swing.plaf.basic.BasicButtonUI;  
  
public 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.graphicsCard=builder.graphicsCard;  
 this.storage=builder.storage;  
 }  
  
 public String toString() {  
 return "CPU: "+CPU+"\nRAM: "+RAM+"\nGraphics Card: "+graphicsCard+"\nStorage: "+storage;  
 }  
  
 public static class Builder{  
 private String CPU;  
 private 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);  
 }  
 }  
}



**4. AdapterPatternExample**

**Gpay.java**

public class Gpay {  
 public void makePayment(double amount)  
 {  
 System.out.println("Gpay processed: "+amount);  
 }  
}

**GpayAdapter.java**

public class GpayAdapter implements PaymentProcessor {  
 Gpay gpay;  
 GpayAdapter(Gpay gpay) {  
 this.gpay=gpay;  
 }  
 @Override  
 public void processorPayment(double amt) {  
 gpay.makePayment(amt);  
 }  
}

**Main.java**  
public class Main {  
 public static void main(String[] args) {  
 Gpay gpay=new Gpay();  
 gpay.makePayment(20000);  
 PaymentProcessor pay=new GpayAdapter(gpay);  
  
 PayPal paypal=new PayPal();  
 paypal.sendPayment(568000.31);  
 PaymentProcessor pay1=new PayPalAdapter(paypal);  
  
 }  
}

**PaymentProcessor.java**

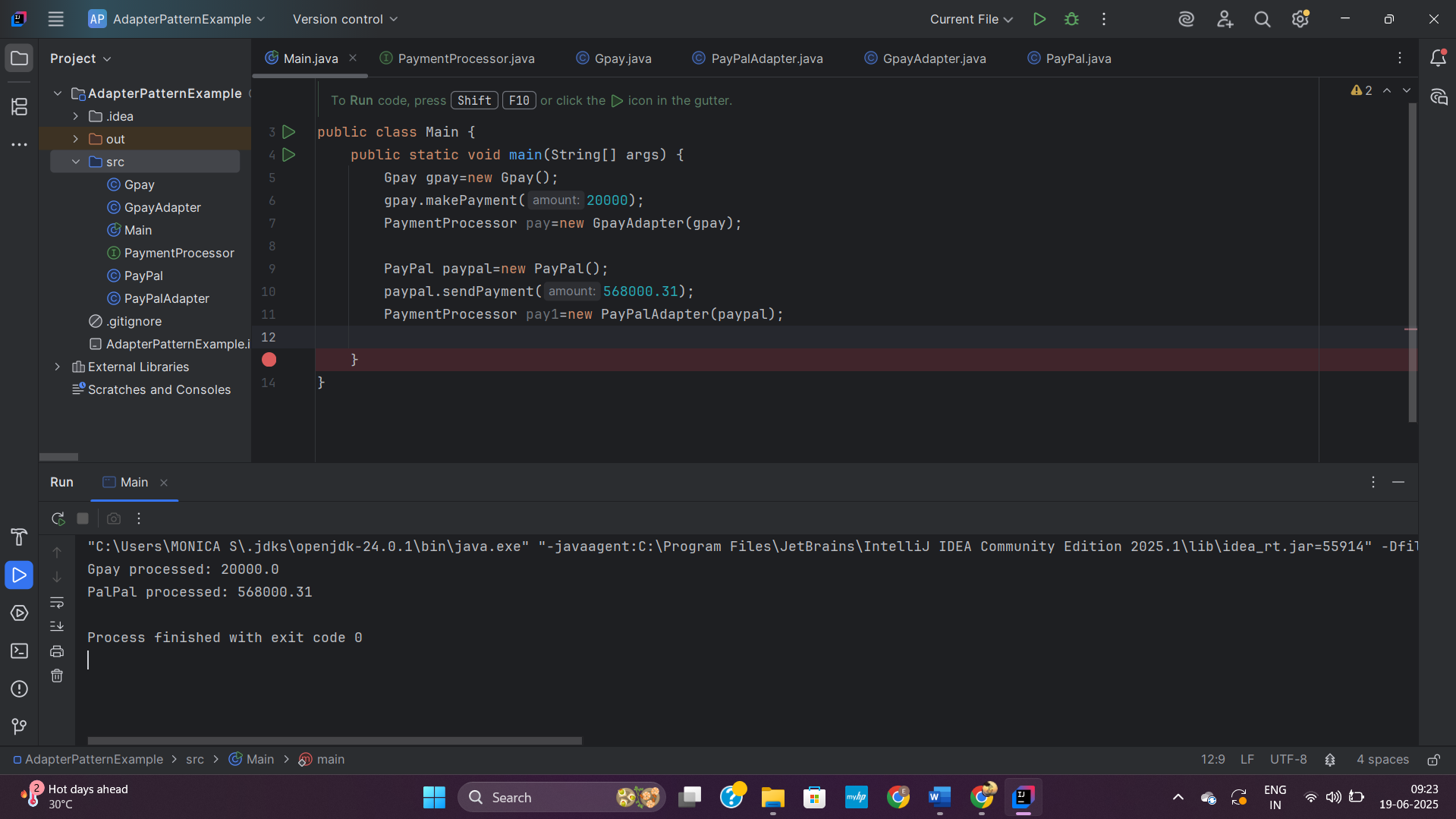
public interface PaymentProcessor {  
 void processorPayment(double amt);  
}

**PayPal.java**

public class PayPal {  
 public void sendPayment(double amount) {  
 System.out.println("PalPal processed: "+amount);  
 }  
}

**PayPalAdapter.java**

public class PayPalAdapter implements PaymentProcessor{  
 PayPal paypal;  
 public PayPalAdapter(PayPal payPal) {  
 this.paypal=payPal;  
 }  
 @Override  
 public void processorPayment(double amt) {  
 paypal.sendPayment(amt);  
 }  
}



**5. DecoratorPatternExample**

**EmailNotifier.java**

public class EmailNotifier implements Notifier{  
 @Override  
 public void send(String message) {  
 System.out.println("Email sent: "+message);  
 }  
}

**Main.java**  
public class Main {  
 public static void main(String[] args) {  
 Notifier notifier = new EmailNotifier();  
 notifier = new SMSNotifierDecorator(notifier);  
 notifier = new SlackNotifierDecorator(notifier);  
 notifier.send("Server is down! Immediate action required.");  
 }  
}

**Notifier.java**

public interface Notifier {  
 void send(String message);  
  
}

**NotifierDecoratoer.java**

public abstract class NotifierDecorator implements Notifier{  
 protected Notifier notifiy;  
 public NotifierDecorator(Notifier notifier) {  
 this.notifiy=notifier;  
 }  
 @Override  
 public void send(String message) {  
 notifiy.send(message);  
 }  
}

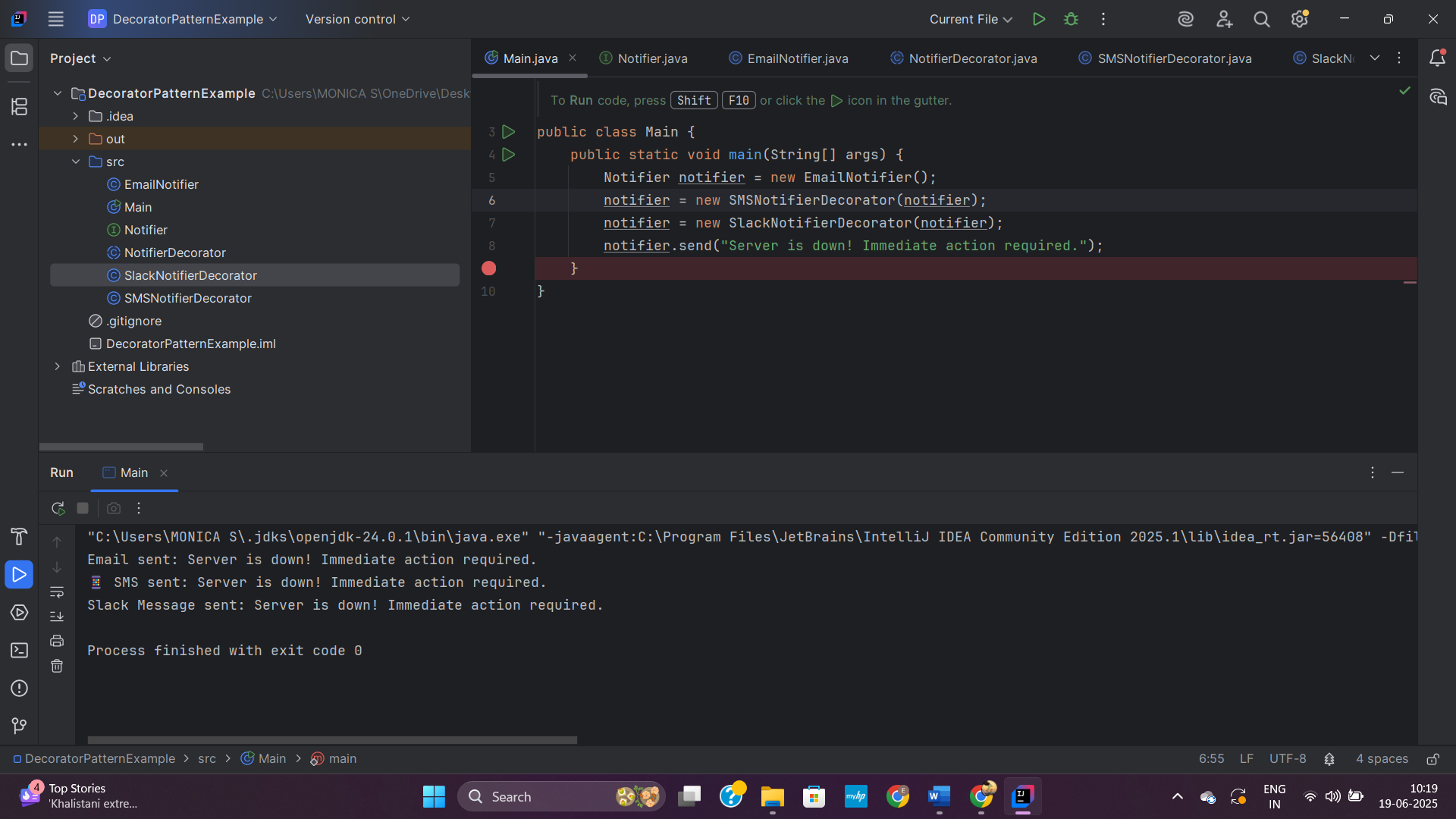
**SlackNotifierDecorator.java**

public class SlackNotifierDecorator extends NotifierDecorator{  
 protected Notifier notify;  
 public SlackNotifierDecorator(Notifier notifier) {  
 super(notifier);  
 }  
 public void send(String message) {  
 super.send(message);  
 System.out.println("Slack Message sent: "+message);  
 }  
}

**SMSNotifierDecorator.java**

public class SMSNotifierDecorator extends NotifierDecorator{  
  
 public SMSNotifierDecorator(Notifier notifier) {  
 super(notifier);  
 }  
 @Override  
 public void send(String message) {  
 super.send(message);

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



**6. ProxyPatternExample**

**Image.java**

public interface Image {  
 public void display();  
}

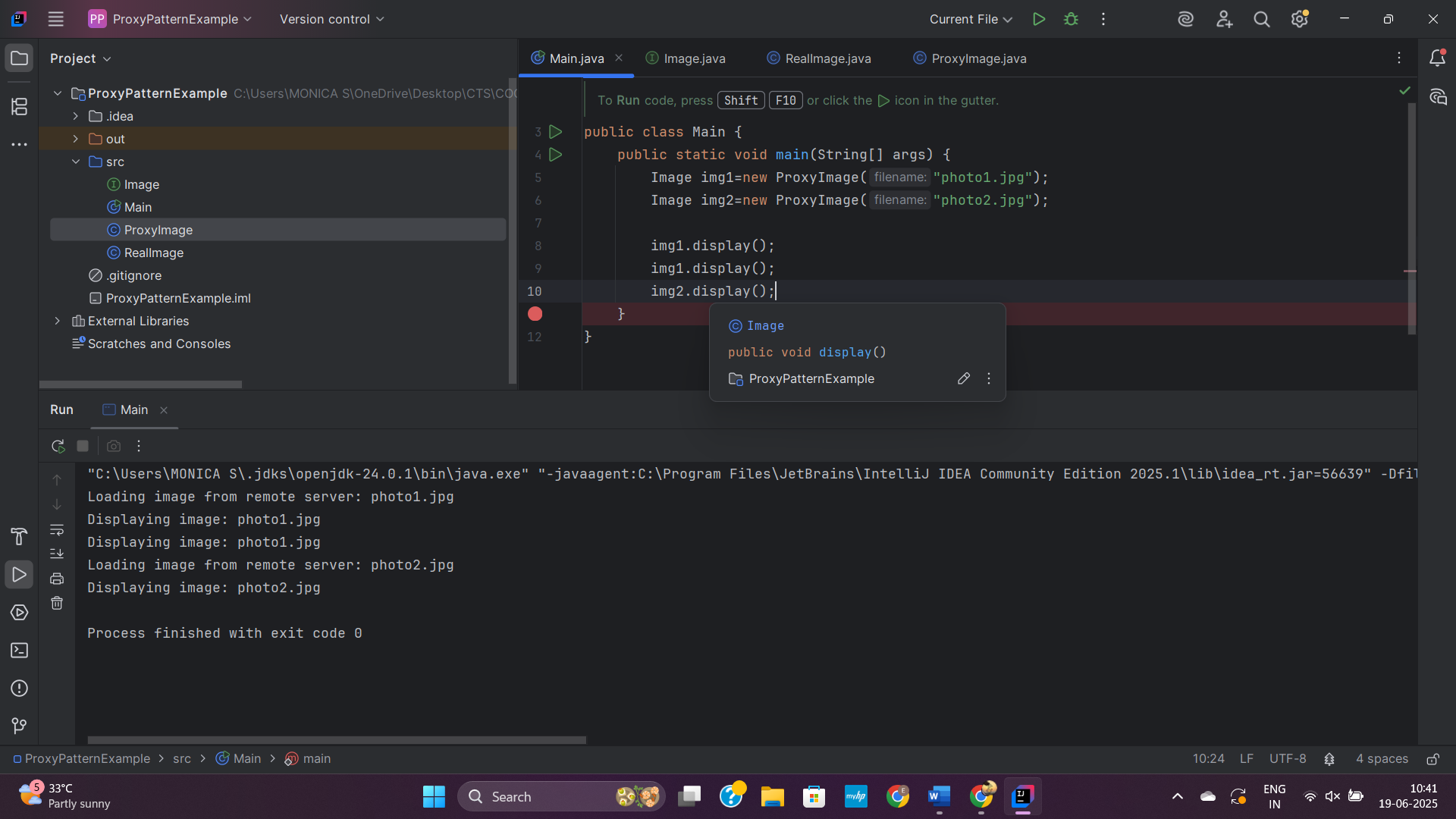
**Main.java**  
public class Main {  
 public static void main(String[] args) {  
 Image img1=new ProxyImage("photo1.jpg");  
 Image img2=new ProxyImage("photo2.jpg");  
  
 img1.display();  
 img1.display();  
 img2.display();  
 }  
}

**ProxyImage.java**

public class ProxyImage implements Image{  
 private String filename;  
 private RealImage real;  
  
 public ProxyImage(String filename) {  
 this.filename=filename;  
 }  
 @Override  
 public void display() {  
 if(real==null) {  
 real=new RealImage(filename);  
 }  
 real.display();  
 }  
}

**RealImage.java**

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);  
 }  
}

****

**7. ObserverPatternExample**

**Main.java**

public class Main {  
 public static void main(String[] args) {  
 Observer m=new MobileApp("John");  
 Observer w=new WebApp("Almend");  
 StockMarket stockMarket=new StockMarket();  
 stockMarket.register(m);  
 stockMarket.register(w);  
 stockMarket.stockPrice(15896.68);  
 stockMarket.deRegister(m);  
 }  
}

**MobileApp.java**

public class MobileApp implements Observer{  
 String name;  
 public MobileApp(String name) {  
 this.name=name;  
 }  
 @Override  
 public void update(double price) {  
 System.out.println("Stock Price updated: "+price);  
 }  
}

**Observer.java**

public interface Observer {  
 public void update(double price);  
}

**Stock.java**

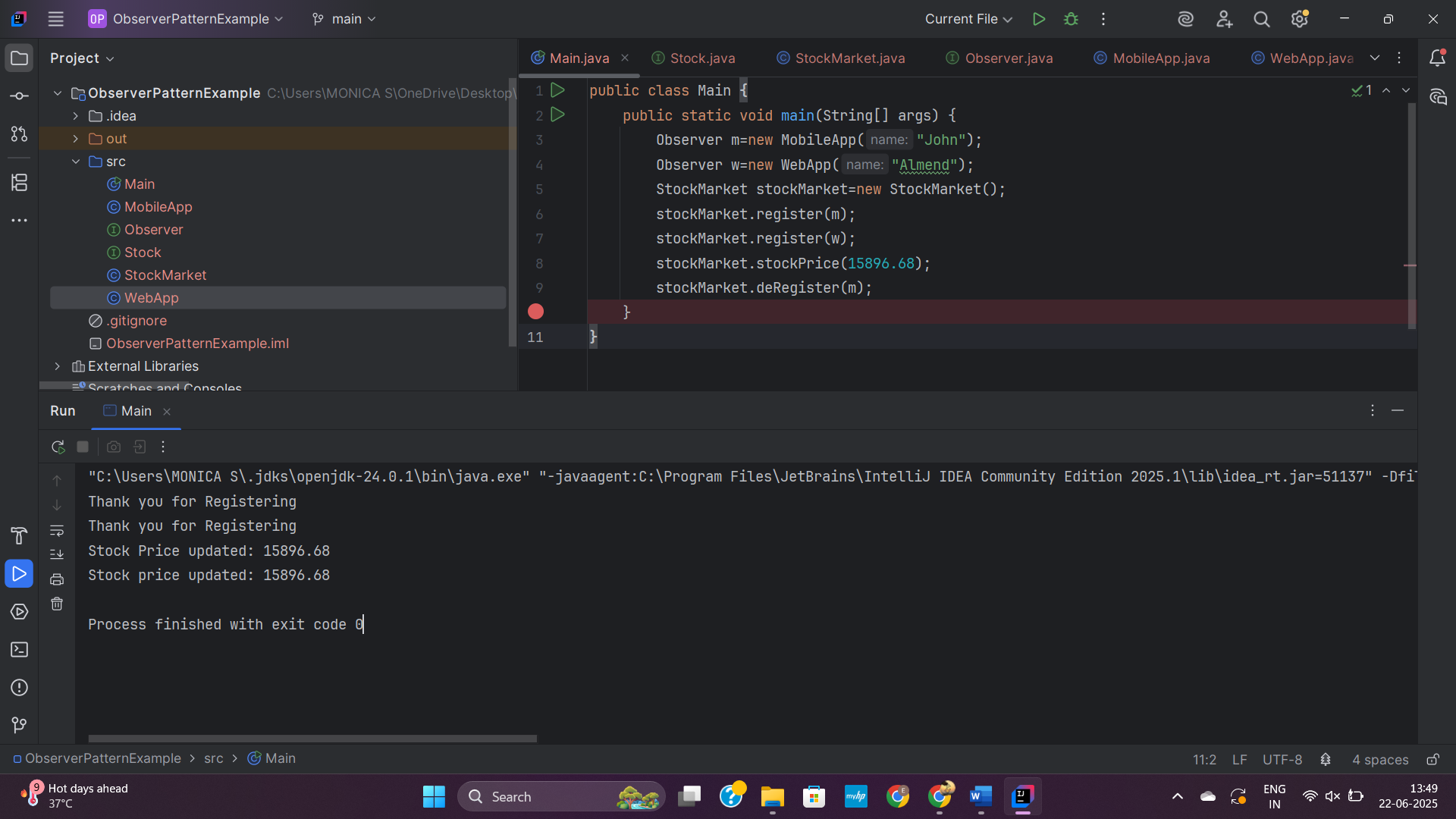
public interface Stock {  
 public void register(Observer o);  
 public void deRegister(Observer o);  
 public void notifyObserver();  
}

**StockMarket.java**

import java.util.ArrayList;  
import java.util.List;  
  
public class StockMarket implements Stock{  
 double price;  
 List<Observer> observer=new ArrayList<>();  
 @Override  
 public void register(Observer o) {  
 System.out.println("Thank you for Registering");  
 observer.add(o);  
 }  
  
 @Override  
 public void deRegister(Observer o) {  
 observer.remove(o);  
 }  
  
 @Override  
 public void notifyObserver() {  
 for (Observer o: observer) {  
 o.update(price);  
 }  
 }  
  
 public void stockPrice(double price){  
 this.price=price;  
 notifyObserver();  
 }  
}

**WebApp.java**

public class WebApp implements Observer{  
 String name;  
 public WebApp(String name) {  
 this.name=name;  
 }  
 @Override  
 public void update(double price) {  
 System.out.println("Stock price updated: "+price);  
 }  
}



**8. StrategyPatternExample**

**CreditCardPayment.java**

public class CreditCardPayment implements PaymentStrategy{  
 String cardNumber;  
 public CreditCardPayment(String cardNumber) {  
 this.cardNumber=cardNumber;  
 }  
 @Override  
 public void pay(double amt) {  
 System.out.println("Paid "+amt+" using Credit card number: "+cardNumber);  
 }  
}

**Main.java**  
public class Main {  
 public static void main(String[] args) {  
 PaymentContext c = new PaymentContext();  
 PaymentStrategy creditCard = new CreditCardPayment("1234-5678-9012-3456");  
 c.setPaymentStrategy(creditCard);  
 c.payAmount(15200.00);  
 PaymentStrategy paypal = new PayPalPayment("user@gmail.com");  
 c.setPaymentStrategy(paypal);  
 c.payAmount(19990.99);  
 }  
}

**PaymentContext.java**

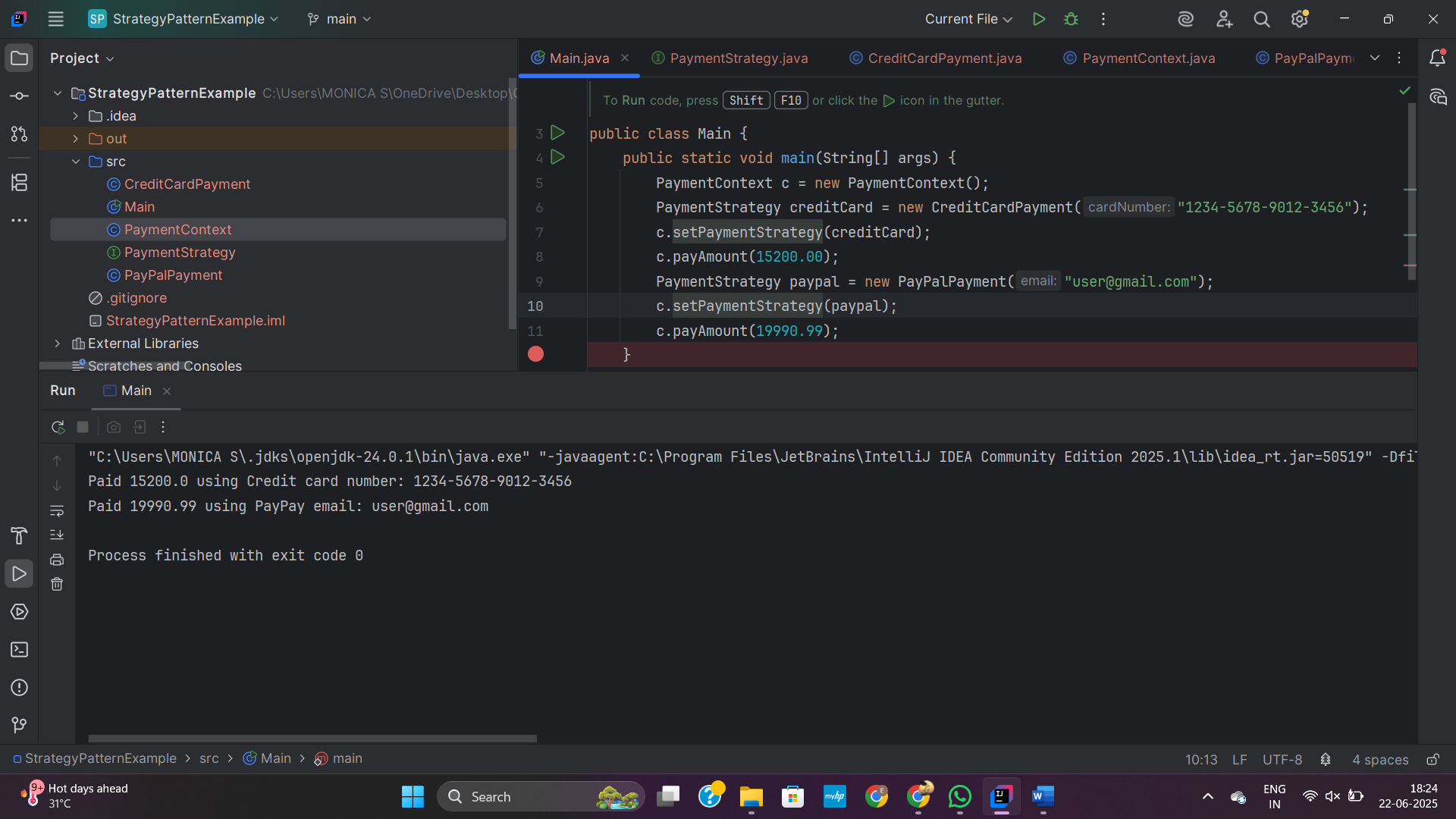
public class PaymentContext {  
 PaymentStrategy strategy;  
 public void setPaymentStrategy(PaymentStrategy strategy) {  
 this.strategy=strategy;  
 }  
 public void payAmount(double amount) {  
 if (strategy == null) {  
 System.out.println("No payment method selected!");  
 } else {  
 strategy.pay(amount);  
 }  
 }  
}

**PaymentStrategy.java**

public interface PaymentStrategy {  
 void pay(double amt);  
}

**PayPalPayment.java**

public class PayPalPayment implements PaymentStrategy{  
 String email;  
 public PayPalPayment(String email) {  
 this.email=email;  
 }  
 @Override  
 public void pay(double amt) {  
 System.out.println("Paid "+amt+" using PayPay email: "+email);  
  
 }  
}



**9. CommandPatternExample**

**Command.java**

public interface Command {  
 void execute();  
}

**Light.java**

public class Light {  
 void turnOn(){  
 System.out.println("Light Turned On");  
 }  
 void turnOff(){  
 System.out.println("Light Turned OFF");  
 }  
}

**LightOffCommand.java**

public class LightOffCommand implements Command{  
 Light light;  
 public LightOffCommand(Light light) {  
 this.light=light;  
 }  
 @Override  
 public void execute() {  
 light.turnOff();  
 }  
}

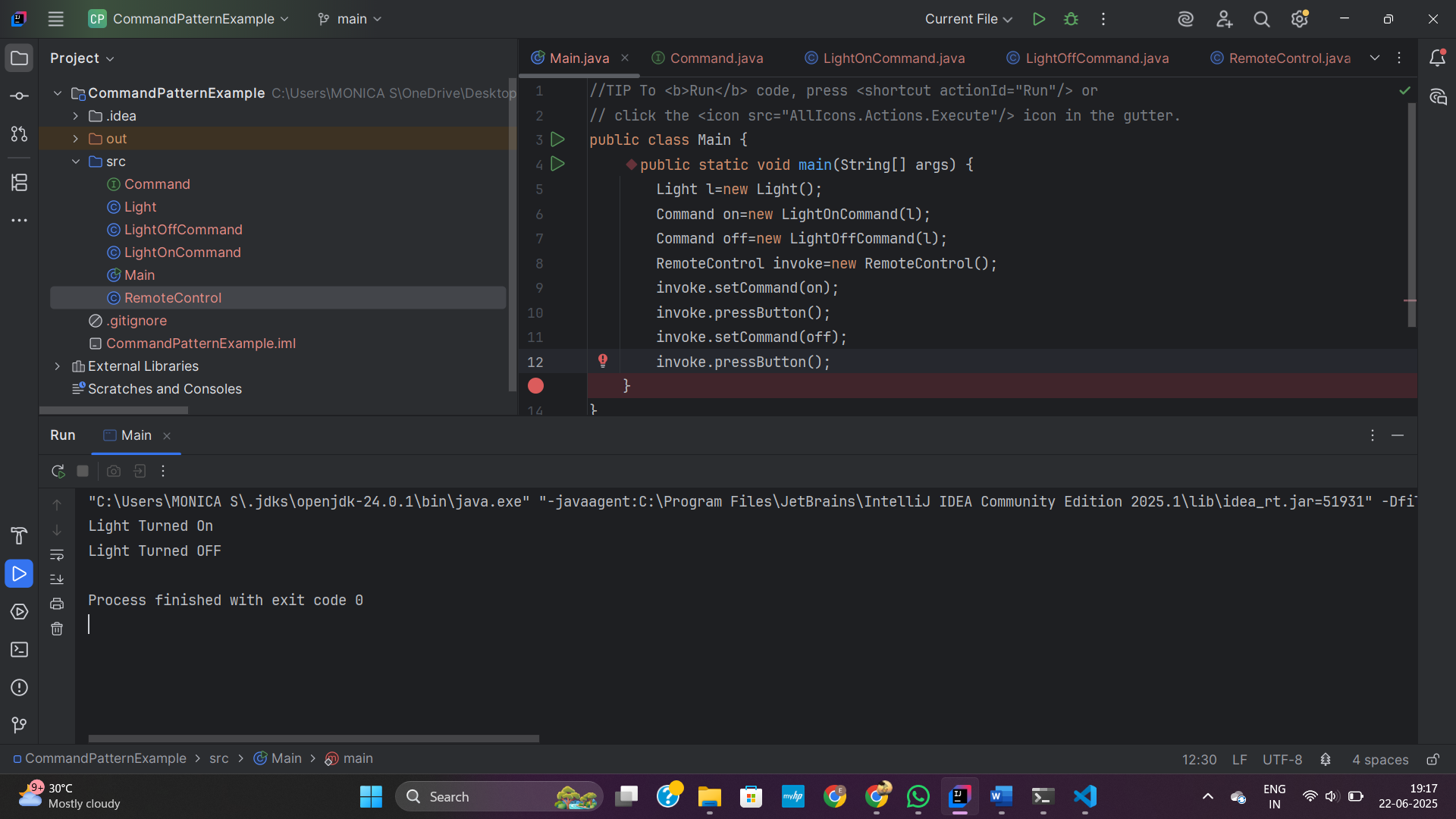
**LigitOnCommand.java**

public class LightOnCommand implements Command{  
 Light light;  
 public LightOnCommand(Light light) {  
 this.light=light;  
 }  
 @Override  
 public void execute() {  
 light.turnOn();  
 }  
}

**Main.java**  
public class Main {  
 public static void main(String[] args) {  
 Light l=new Light();  
 Command on=new LightOnCommand(l);  
 Command off=new LightOffCommand(l);  
 RemoteControl invoke=new RemoteControl();  
 invoke.setCommand(on);  
 invoke.pressButton();  
 invoke.setCommand(off);  
 invoke.pressButton();  
 }  
}

**RemoteControl.java**

public class RemoteControl {  
 Command cmd;  
 public void setCommand(Command cmd) {  
 this.cmd=cmd;  
 }  
  
 public void pressButton() {  
 cmd.execute();  
 }  
  
}



**10. MVCPatternExample**

**Main.java**

public class Main {  
 public static void main(String[] args) {  
 Student student = new Student("S101", "John", "A");  
 StudentView view = new StudentView();  
 StudentController controller = new StudentController(student, view);  
 controller.updateView();  
 controller.setStudentName("Jane");  
 controller.setStudentGrade("A+");  
 controller.updateView();  
 }  
}

**Student.java**

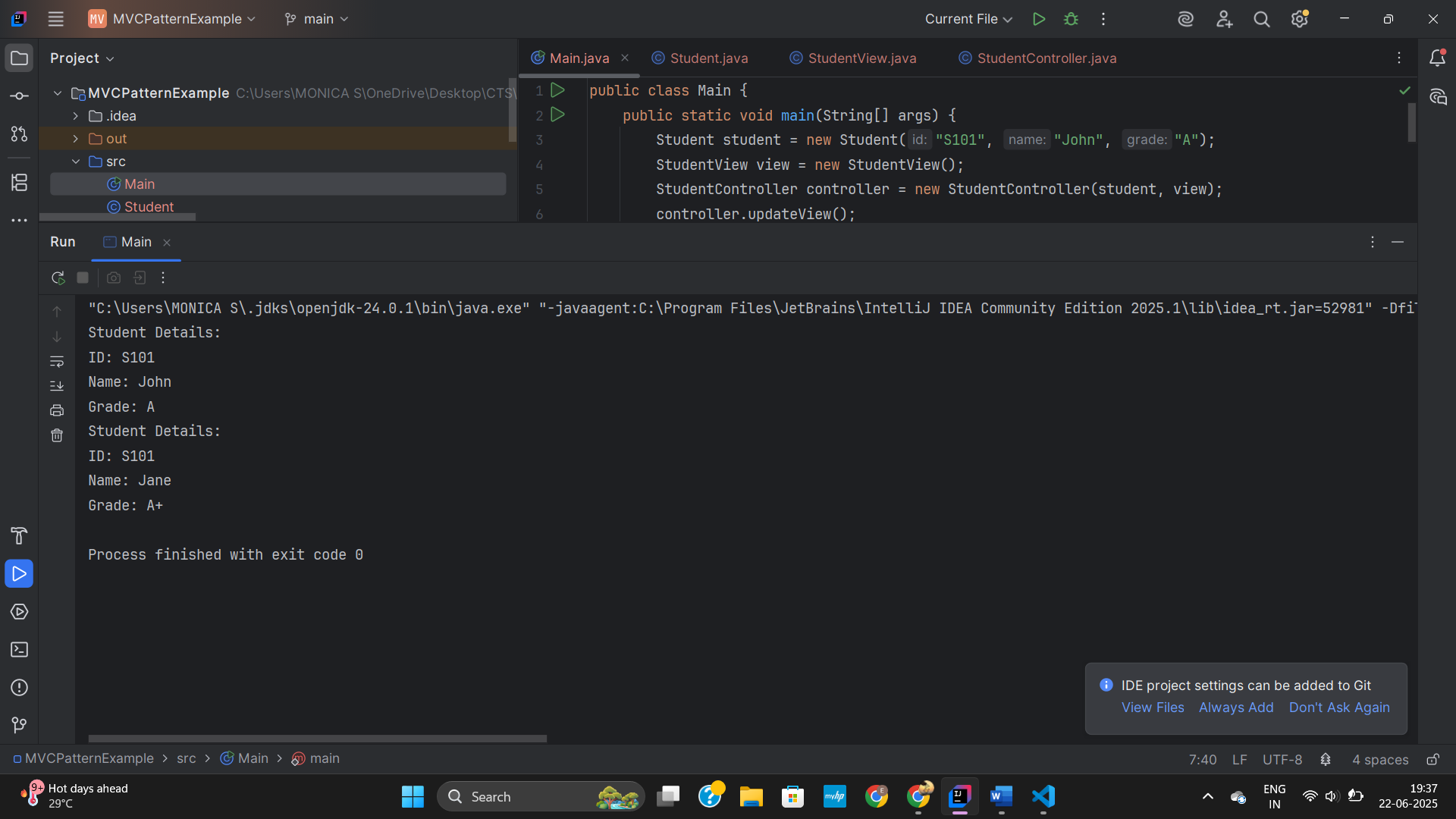
public class Student {  
 private String id;  
 private String name;  
 private String grade;  
 public Student(String id, String name, String grade) {  
 this.id = id;  
 this.name = name;  
 this.grade = grade;  
 }  
 public String getId() {  
 return id;  
 }  
  
 public void setId(String id) {  
 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;  
 }  
}

**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 setStudentGrade(String grade) {  
 model.setGrade(grade);  
 }  
  
 public String getStudentGrade() {  
 return model.getGrade();  
 }  
  
 public void updateView() {  
 view.displayStudentDetails(model.getId(), model.getName(), model.getGrade());  
 }  
}

**StudentView.java**

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);  
 }  
}



**11. DependencyInjectionExample**

**CustomerRepository.java**

public interface CustomerRepository {  
 String findCustomerById(String id);  
}

**CustomerRepositoryImpl.java**

public class CustomerRepositoryImpl implements CustomerRepository {  
 @Override  
 public String findCustomerById(String id) {  
 return "Customer Name for ID " + id;  
 }  
}

**CustomerService.java**

public class CustomerService {  
 private CustomerRepository customerRepository;  
 public CustomerService(CustomerRepository customerRepository) {  
 this.customerRepository = customerRepository;  
 }  
 public void getCustomerDetails(String id) {  
 String customer = customerRepository.findCustomerById(id);  
 System.out.println("Customer Details: " + customer);  
 }  
}

**Main.java**

public class Main {  
 public static void main(String[] args) {  
 CustomerRepository repo = new CustomerRepositoryImpl();  
 CustomerService service = new CustomerService(repo);  
 service.getCustomerDetails("C101");  
 }  
}

