**Week-1 Hands-On Exercises**

**1.Singleton Pattern**

**Logger.java:**

public class Logger {

private static Logger instance;

private Logger() {

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

}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void log(String message) {

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

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

logger1.log("First message");

logger2.log("Second message");

if (logger1 == logger2) {

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

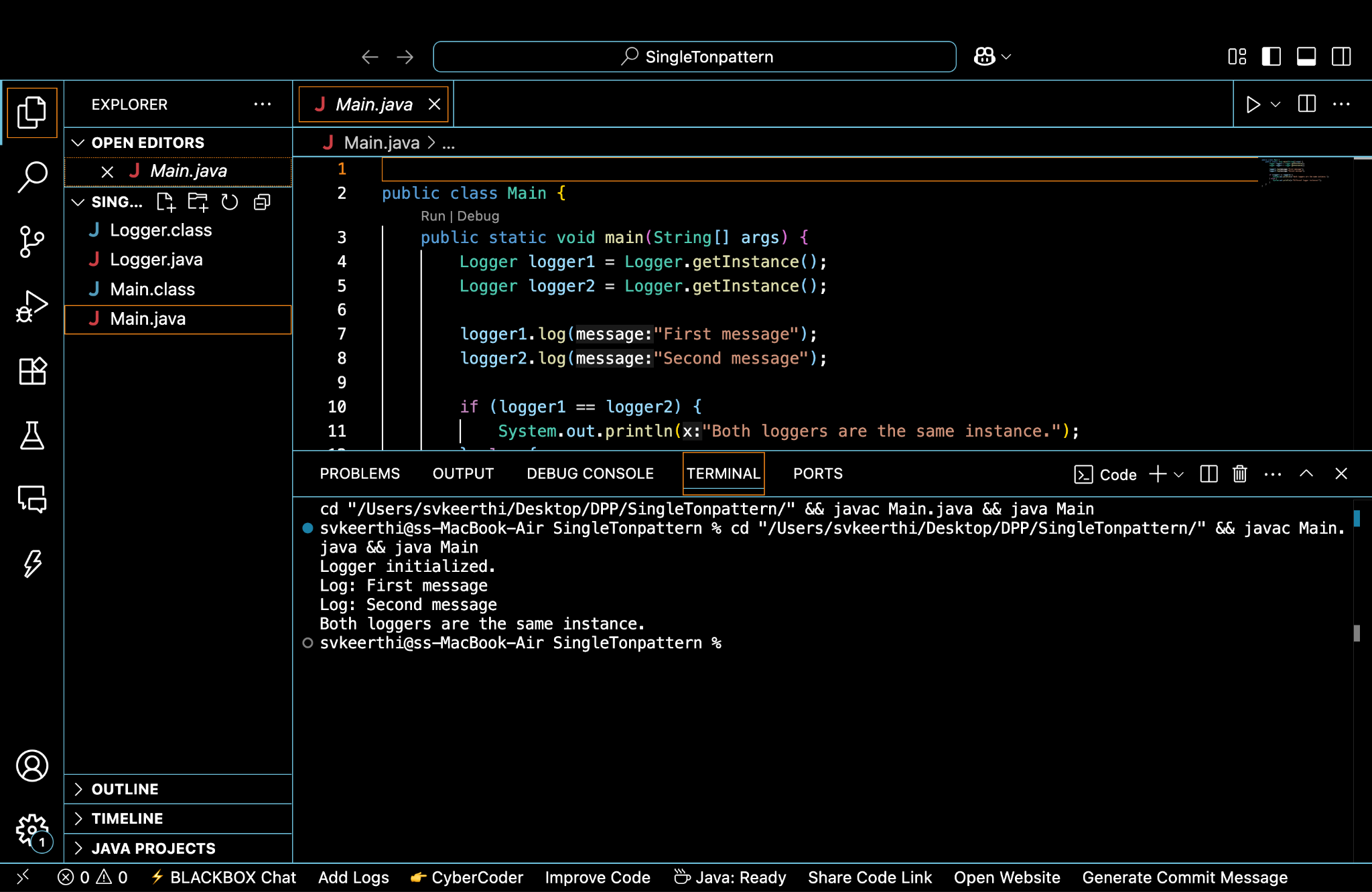
} else {

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

}

}

}

**Output:**

**2.Factory Method Pattern**

public interface Document {

void open();

}

public class WordDocument implements Document {

public void open() {

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

}

}

public class PdfDocument implements Document {

public void open() {

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

}

}

public class ExcelDocument implements Document {

public void open() {

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

}

}

public abstract class DocumentFactory {

public abstract Document createDocument();

}

public class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

public class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

public class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

public class Main {

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();

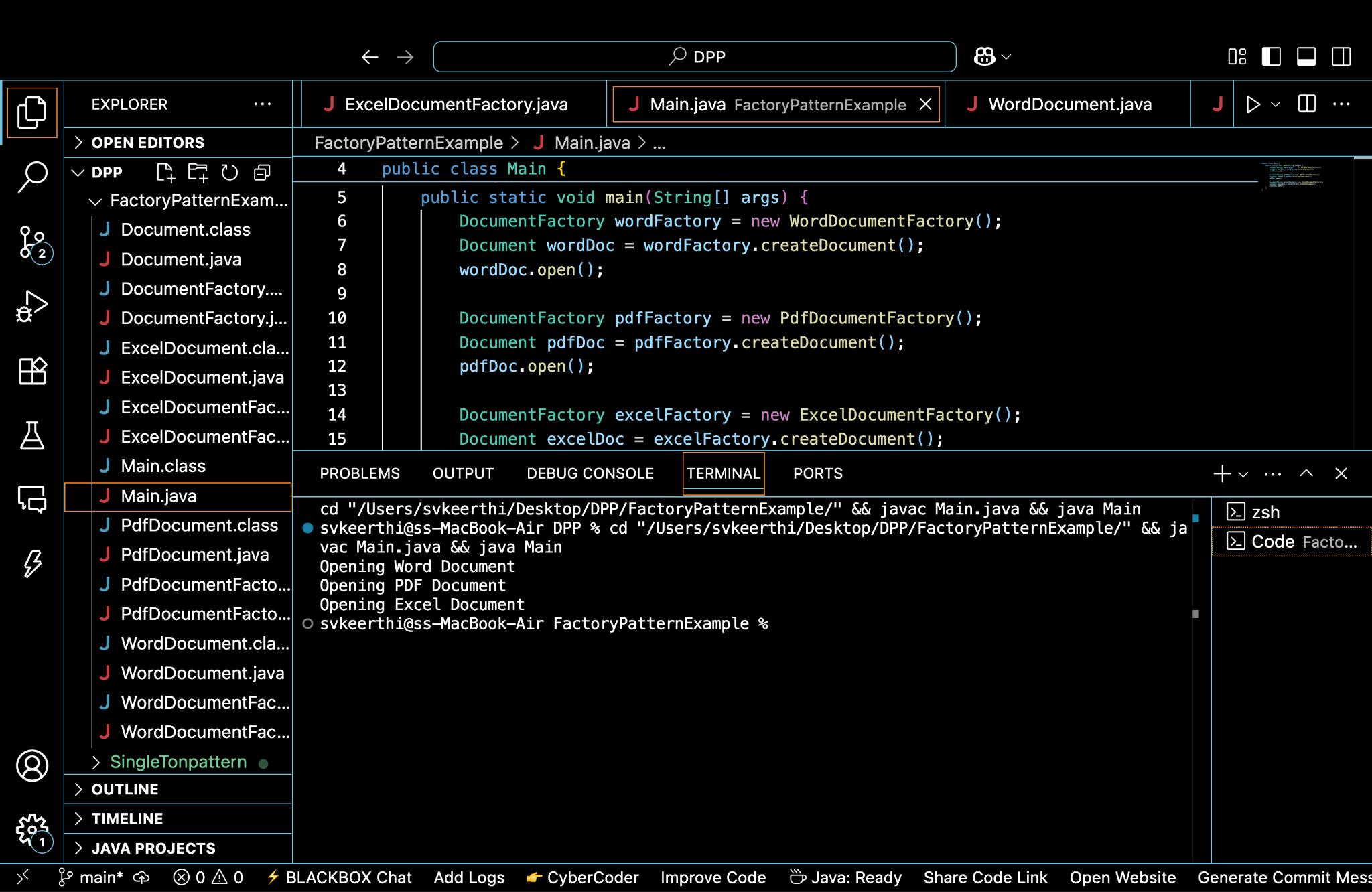
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**Output:**

****

**3.Builder Pattern**

**Computer.java:**

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.storage = builder.storage;

this.graphicsCard = builder.graphicsCard;

}

public void displayConfig() {

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

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

System.out.println("Storage: " + (storage != null ? storage : "Not included"));

System.out.println("Graphics Card: " + (graphicsCard != null ? graphicsCard : "Not included"));

}

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);

}

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

Computer basicComputer = new Computer.Builder("Intel i3", "8GB")

.build();

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

.setStorage("1TB SSD")

.setGraphicsCard("NVIDIA RTX 3080")

.build();

System.out.println("Basic Computer Config:");

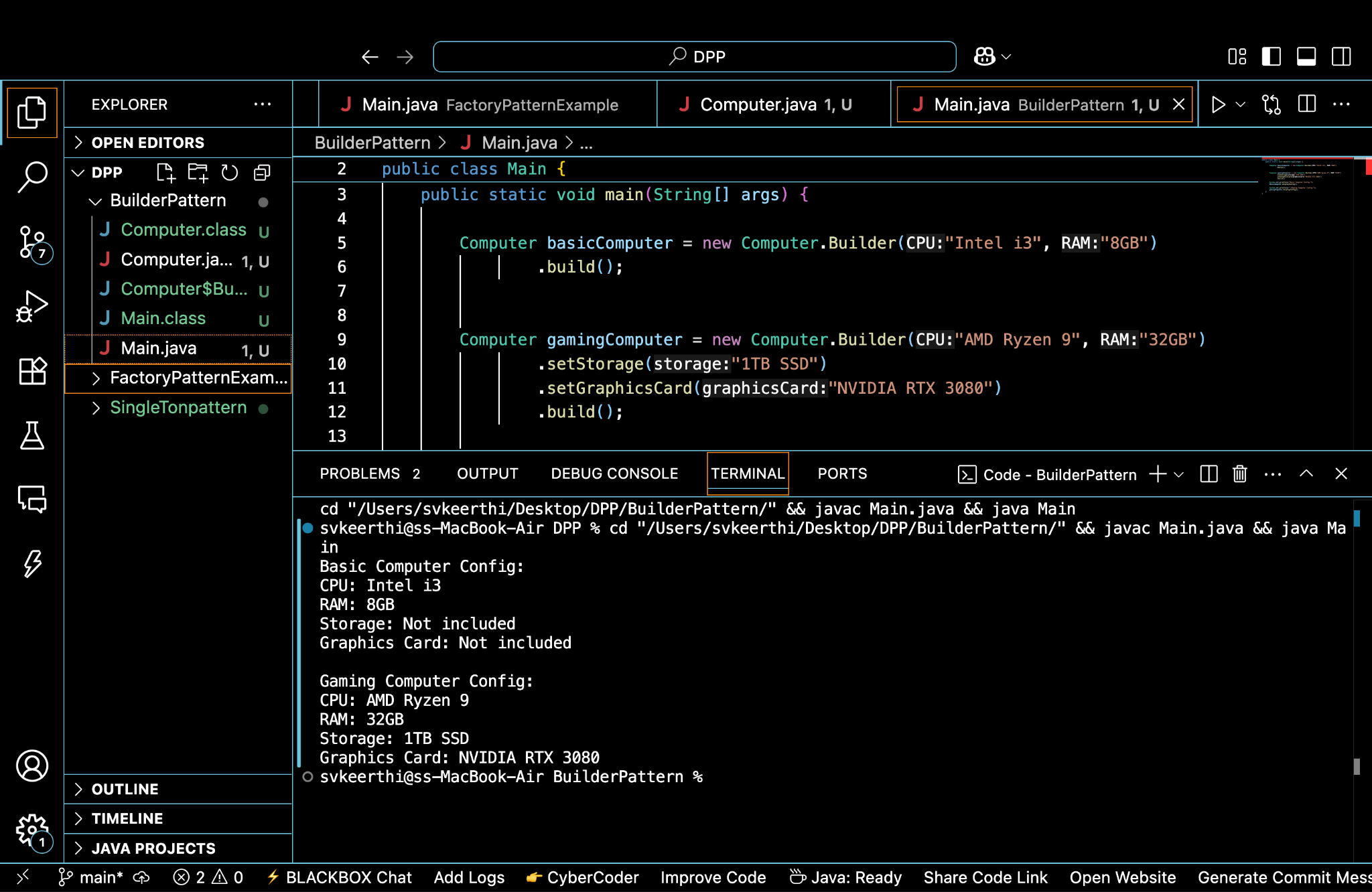
basicComputer.displayConfig();

System.out.println("\nGaming Computer Config:");

gamingComputer.displayConfig();

}

}

**Output:**

**4.Adapter Pattern**

public interface PaymentProcessor {

void processPayment(double amount);

}

public class PayPalGateway {

public void sendPayment(double amount) {

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

}

}

public class StripeGateway {

public void makePayment(double value) {

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

}

}

public class PayPalAdapter implements PaymentProcessor {

private PayPalGateway paypal;

public PayPalAdapter(PayPalGateway paypal) {

this.paypal = paypal;

}

public void processPayment(double amount) {

paypal.sendPayment(amount);

}

}

public class StripeAdapter implements PaymentProcessor {

private StripeGateway stripe;

public StripeAdapter(StripeGateway stripe) {

this.stripe = stripe;

}

public void processPayment(double amount) {

stripe.makePayment(amount);

}

}

public class Main {

public static void main(String[] args) {

PayPalGateway paypal = new PayPalGateway();

PaymentProcessor paypalProcessor = new PayPalAdapter(paypal);

paypalProcessor.processPayment(500.0);

StripeGateway stripe = new StripeGateway();

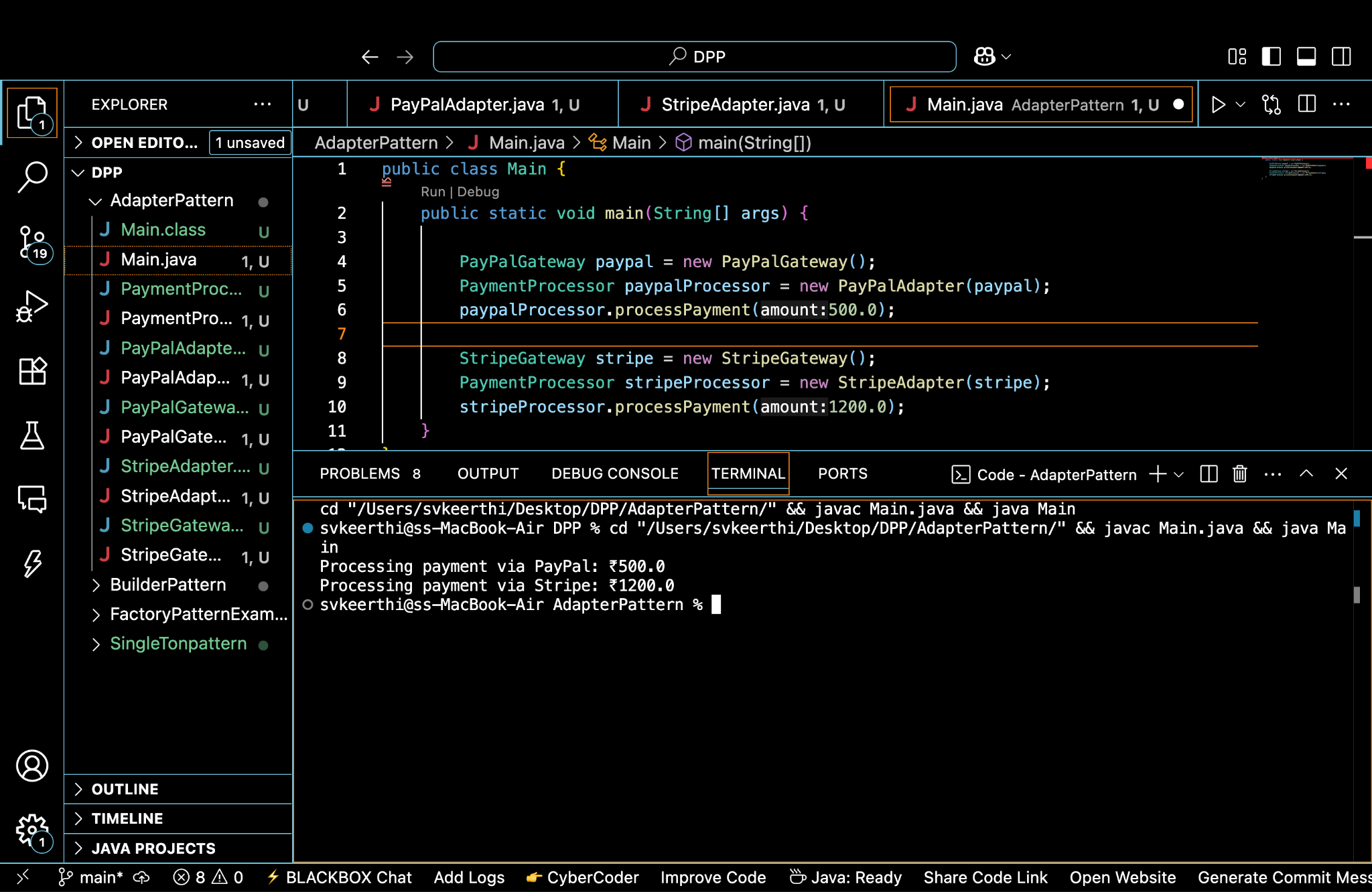
PaymentProcessor stripeProcessor = new StripeAdapter(stripe);

stripeProcessor.processPayment(1200.0);

}

}

**Output:**

****

**5.Decorator Pattern**

public interface Notifier {

void send(String message);

}

public class EmailNotifier implements Notifier {

public void send(String message) {

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

}

}

public abstract class NotifierDecorator implements Notifier {

protected Notifier wrappedNotifier;

public NotifierDecorator(Notifier notifier) {

this.wrappedNotifier = notifier;

}

public void send(String message) {

wrappedNotifier.send(message); // Delegate to wrapped notifier

}

}

public class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message); // Send previous notifications

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

}

}

public class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message); // Send previous notifications

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

}

}

public class Main {

public static void main(String[] args) {

Notifier basicNotifier = new EmailNotifier();

Notifier smsNotifier = new SMSNotifierDecorator(basicNotifier);

Notifier multiNotifier = new SlackNotifierDecorator(smsNotifier);

multiNotifier.send("System alert: High memory usage");

}

}

**Output:**

****

**6. Proxy Pattern**

public interface Image {

void display();

}

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 server: " + fileName);

}

public void display() {

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

}

}

public 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); // Load only once

} else {

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

}

realImage.display();

}

}

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();

}

}

****

**7.Observer Pattern**

public interface Observer {

void update(String stockName, double newPrice);

}

public interface Stock {

void registerObserver(Observer o);

void removeObserver(Observer o);

void notifyObservers();

}

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 setStock(String name, double price) {

this.stockName = name;

this.stockPrice = price;

notifyObservers();

}

public void registerObserver(Observer o) {

observers.add(o);

}

public void removeObserver(Observer o) {

observers.remove(o);

}

public void notifyObservers() {

for (Observer o : observers) {

o.update(stockName, stockPrice);

}

}

}

public class MobileApp implements Observer {

public void update(String stockName, double newPrice) {

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

}

}

public class WebApp implements Observer {

public void update(String stockName, double newPrice) {

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

}

}

public class Main {

public static void main(String[] args) {

StockMarket market = new StockMarket();

Observer mobile = new MobileApp();

Observer web = new WebApp();

market.registerObserver(mobile);

market.registerObserver(web);

market.setStock("INFY", 1495.50);

market.setStock("TCS", 3599.00);

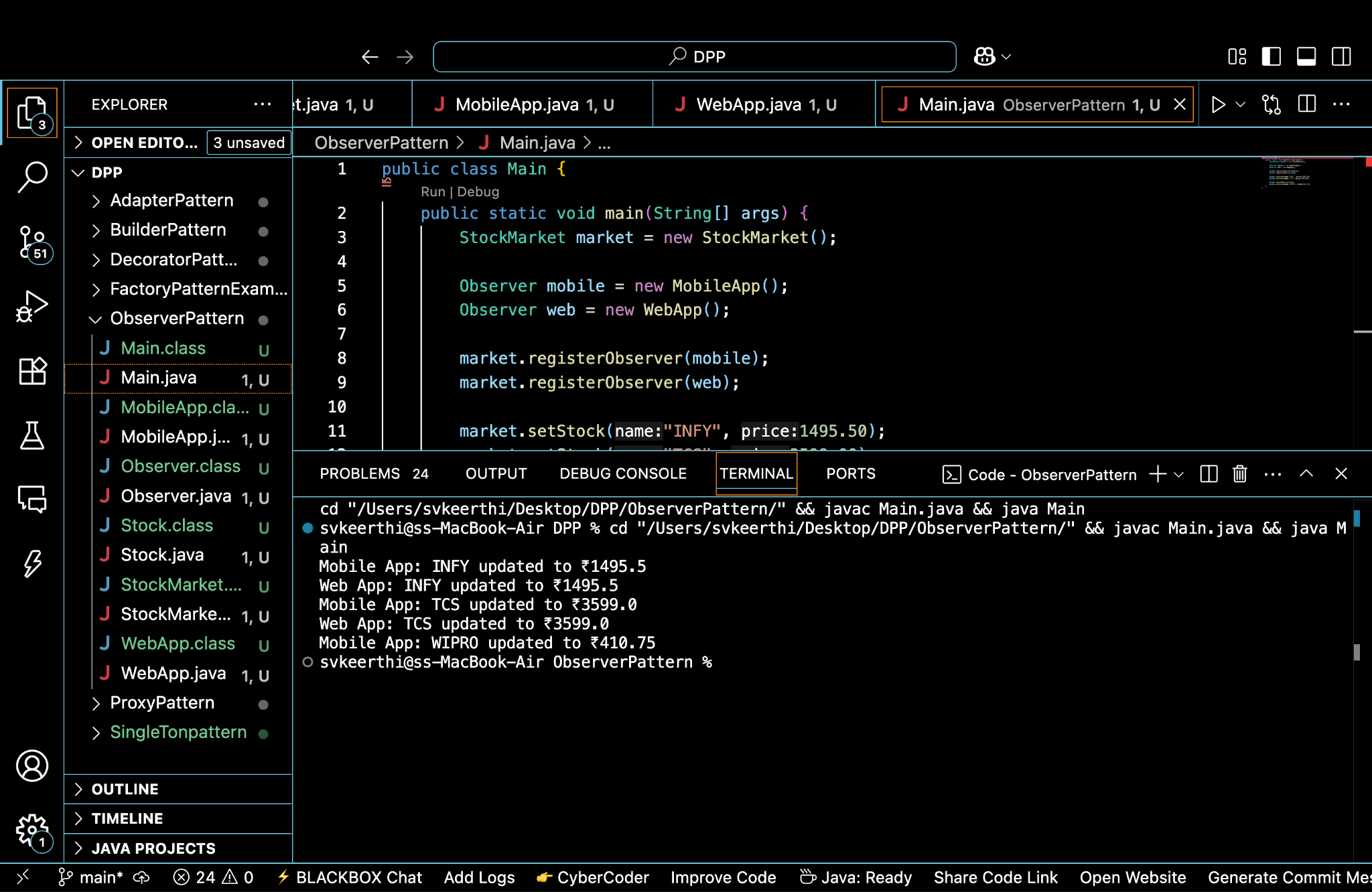
market.removeObserver(web);

market.setStock("WIPRO", 410.75);

}

}

**Output:**

****

**8. Strategy Pattern**

public interface PaymentStrategy {

void pay(double amount);

}

public class CreditCardPayment implements PaymentStrategy {

private String cardNumber;

public CreditCardPayment(String cardNumber) {

this.cardNumber = cardNumber;

}

public void pay(double amount) {

System.out.println("Paid ₹" + amount + " using Credit Card ending in " + cardNumber.substring(cardNumber.length() - 4));

}

}

public class PayPalPayment implements PaymentStrategy {

private String email;

public PayPalPayment(String email) {

this.email = email;

}

public void pay(double amount) {

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

}

}

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("No payment method selected.");

} else {

strategy.pay(amount);

}

}

}

public class Main {

public static void main(String[] args) {

PaymentContext context = new PaymentContext();

context.setPaymentStrategy(new CreditCardPayment("1234567890123456"));

context.processPayment(2500.00);

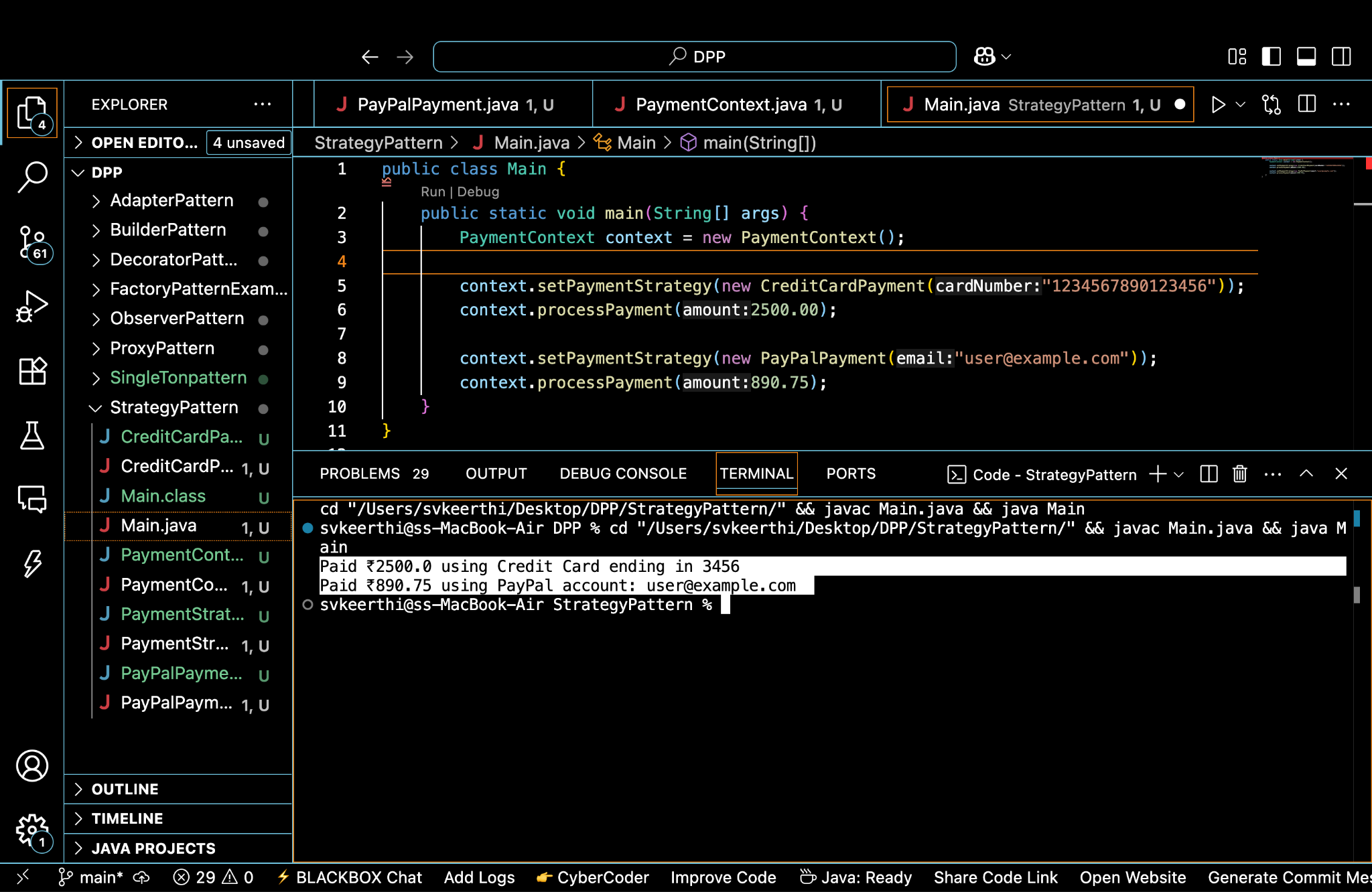
context.setPaymentStrategy(new PayPalPayment("user@example.com"));

context.processPayment(890.75);

}

}

**Output:**

****

**9.Command Pattern**

public interface Command {

void execute();

}

public class Light {

public void turnOn() {

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

}

public void turnOff() {

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

}

}

public class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOn();

}

}

public class LightOffCommand implements Command {

private Light light;

public LightOffCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOff();

}

}

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.");

}

}

}

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();

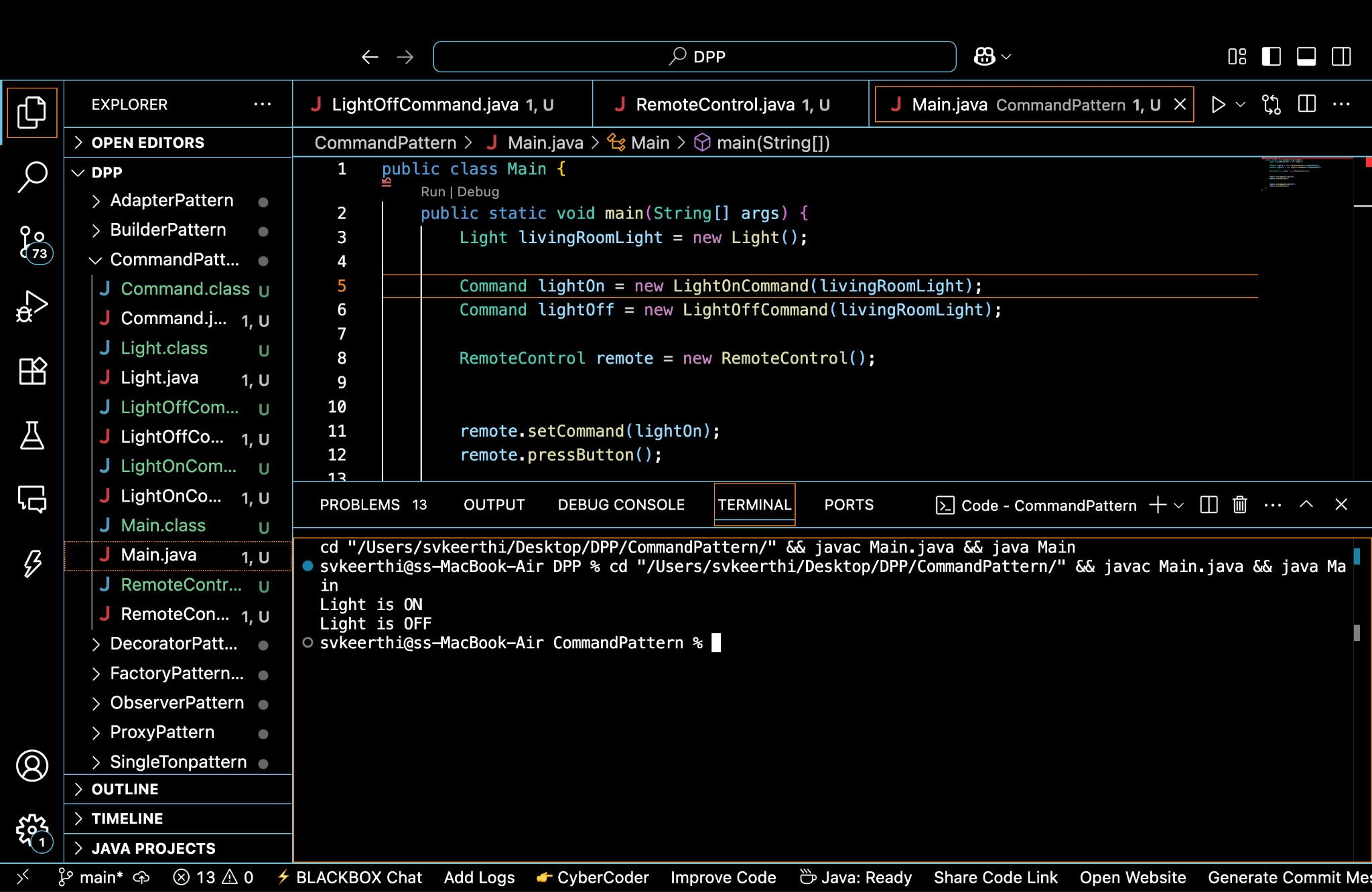
remote.setCommand(lightOff);

remote.pressButton();

}

}

**Output:**

****

**10.MVC Pattern**

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;

}

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;

}

}

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);

}

}

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 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 class Main {

public static void main(String[] args) {

Student student = new Student("Keerthi", "STU1024", "A");

StudentView view = new StudentView();

StudentController controller = new StudentController(student, view);

controller.updateView();

controller.setStudentName("Keerthi V");

controller.setStudentGrade("A+");

System.out.println("\nAfter Update:");

controller.updateView();

}}

**Output:**

****

**11.Dependency Injection**

public interface CustomerRepository {

String findCustomerById(String customerId);

}

public class CustomerRepositoryImpl implements CustomerRepository {

public String findCustomerById(String customerId) {

return "Customer[ID=" + customerId + ", Name=Keerthi, Email=keerthi@example.com]";

}

}

public class CustomerService {

private CustomerRepository repository;

public CustomerService(CustomerRepository repository) {

this.repository = repository;

}

public void showCustomer(String customerId) {

String customerDetails = repository.findCustomerById(customerId);

System.out.println("Customer Details: " + customerDetails);

}

}

public class Main {

public static void main(String[] args) {

CustomerRepository repo = new CustomerRepositoryImpl();

CustomerService service = new CustomerService(repo);

service.showCustomer("CUST1001");

}

}

**Output:**

