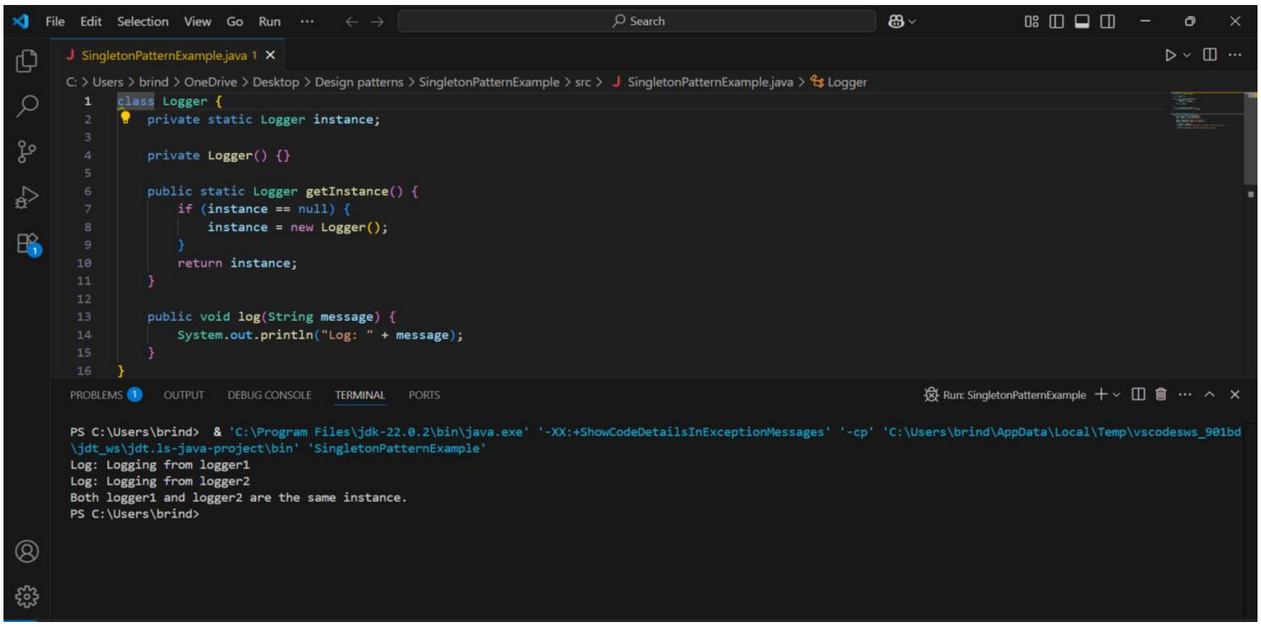
DESIGN PRINCIPLES AND DESIGN PATTERNS EXERCISES

Exercise 1: Implementing the Singleton Pattern

```java
class Logger {
    private static Logger instance;

    private Logger() {}

    public static Logger getInstance() {
        if (instance == null) {
            instance = new Logger();
        }
        return instance;
    }

    public void log(String message) {
        System.out.println("Log: " + message);
    }
}

public class SingletonPatternExample {
    public static void main(String[] args) {
        Logger logger1 = Logger.getInstance();
        Logger logger2 = Logger.getInstance();

        logger1.log("Logging from logger1");
        logger2.log("Logging from logger2");

        if (logger1 == logger2) {
            System.out.println("Both logger1 and logger2 are the same instance.");
        } else {
            System.out.println("Different instances were created.");
        }
    }
}
```

OUTPUT:


File Edit Selection View Go Run ···
Search
SingletonPatternExample.java 1
C: > Users > brind > OneDrive > Desktop > Design patterns > SingletonPatternExample > src > SingletonPatternExample.java > Logger
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Run: SingletonPatternExample

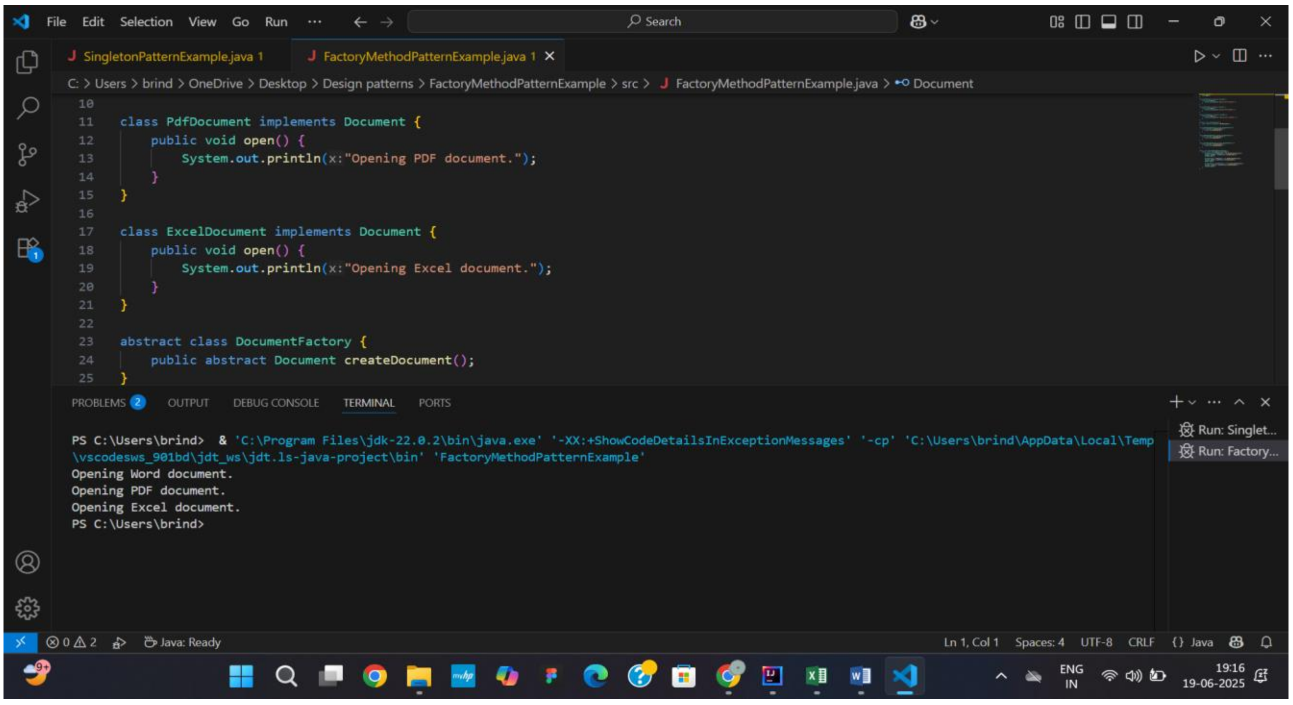
```java
class Logger {
    private static Logger instance;

    private Logger() {}

    public static Logger getInstance() {
        if (instance == null) {
            instance = new Logger();
        }
        return instance;
    }

    public void log(String message) {
        System.out.println("Log: " + message);
    }
}
```

```
PS C:\Users\brind>  & 'C:\Program Files\jdk-22.0.2\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\brind\AppData\Local\Temp\vscodesws_901bd
\jdt_ws\jdt.ls-java-project\bin' 'SingletonPatternExample'
Log: Logging from logger1
Log: Logging from logger2
Both logger1 and logger2 are the same instance.
PS C:\Users\brind>
```

Exercise 2: Implementing the Factory Method Pattern

```java
interface Document {
    void open();
}

class WordDocument implements Document {
    public void open() {
        System.out.println("Opening Word document.");
    }
}

class PdfDocument implements Document {
    public void open() {
        System.out.println("Opening PDF document.");
    }
}

class ExcelDocument implements Document {
    public void open() {
        System.out.println("Opening Excel document.");
    }
}

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

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

class PdfDocumentFactory extends DocumentFactory {
    public Document createDocument() {
        return new PdfDocument();
    }
}

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

public class FactoryMethodPatternExample {
    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:



C: > Users > brind > OneDrive > Desktop > Design patterns > FactoryMethodPatternExample > src > FactoryMethodPatternExample.java > Document

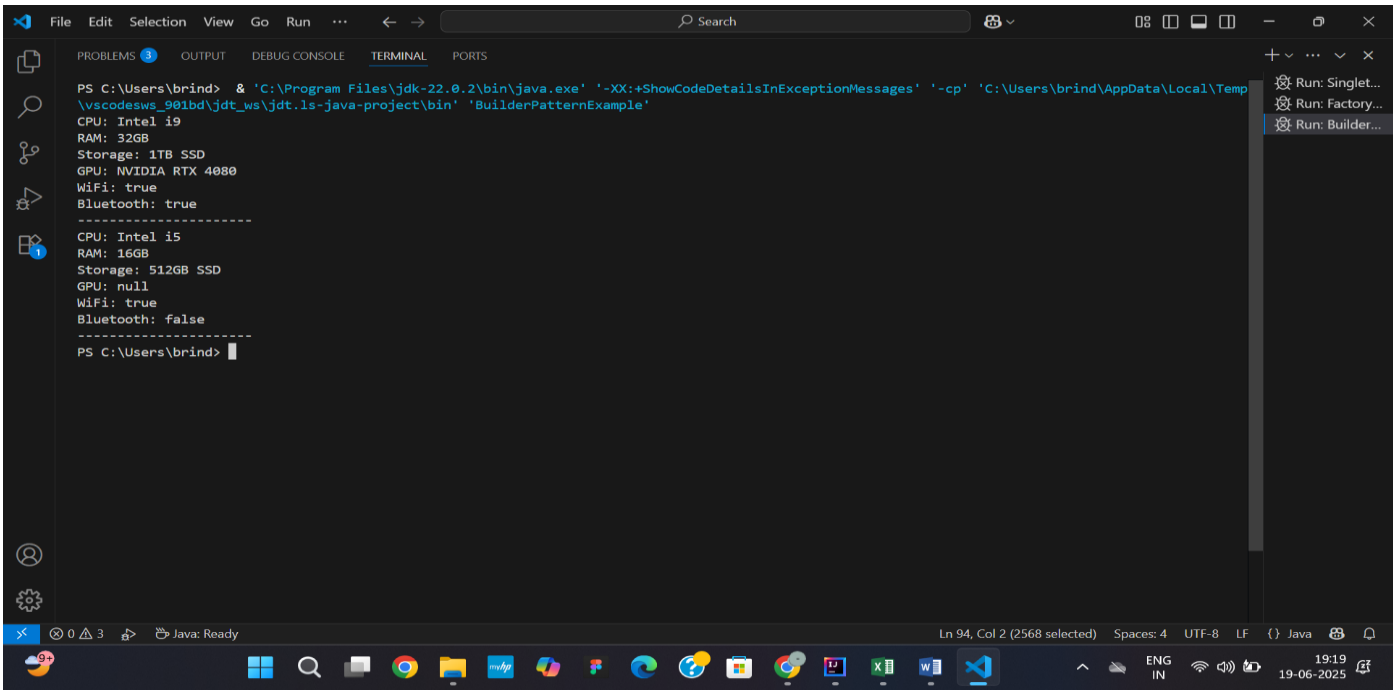
```java

class PdfDocument implements Document {
    public void open() {
        System.out.println(x:"Opening PDF document.");
    }
}

class ExcelDocument implements Document {
    public void open() {
        System.out.println(x:"Opening Excel document.");
    }
}

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

```
PS C:\Users\brind>  & 'C:\Program Files\jdk-22.0.2\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\brind\AppData\Local\Temp
\vscodesws_901bd\jdt_ws\jdt.ls-java-project\bin' 'FactoryMethodPatternExample'
Opening Word document.
Opening PDF document.
Opening Excel document.
PS C:\Users\brind>
```

Exercise 3: Implementing the Builder Pattern

```java
class Computer {
    private String CPU;
    private String RAM;
    private String storage;
    private String GPU;
    private boolean hasWiFi;
    private boolean hasBluetooth;

    private Computer(Builder builder) {
        this.CPU = builder.CPU;
        this.RAM = builder.RAM;
        this.storage = builder.storage;
        this.GPU = builder.GPU;
        this.hasWiFi = builder.hasWiFi;
        this.hasBluetooth = builder.hasBluetooth;
    }

    public void displayConfig() {
        System.out.println("CPU: " + CPU);
        System.out.println("RAM: " + RAM);
        System.out.println("Storage: " + storage);
        System.out.println("GPU: " + GPU);
        System.out.println("WiFi: " + hasWiFi);
        System.out.println("Bluetooth: " + hasBluetooth);
        System.out.println("----------------------");
    }

    public static class Builder {
        private String CPU;
        private String RAM;
        private String storage;
        private String GPU;
        private boolean hasWiFi;
        private boolean hasBluetooth;

        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 setGPU(String GPU) {
            this.GPU = GPU;
            return this;
        }

        public Builder setWiFi(boolean hasWiFi) {
            this.hasWiFi = hasWiFi;
            return this;
        }

        public Builder setBluetooth(boolean hasBluetooth) {
            this.hasBluetooth = hasBluetooth;
            return this;
        }

        public Computer build() {
            return new Computer(this);
        }
    }
}

public class BuilderPatternExample {
    public static void main(String[] args) {
        Computer gamingPC = new Computer.Builder()
                .setCPU("Intel i9")
                .setRAM("32GB")
                .setStorage("1TB SSD")
                .setGPU("NVIDIA RTX 4080")
                .setWiFi(true)
                .setBluetooth(true)
                .build();

        Computer officePC = new Computer.Builder()
                .setCPU("Intel i5")
                .setRAM("16GB")
                .setStorage("512GB SSD")
                .setWiFi(true)
                .setBluetooth(false)
                .build();

        gamingPC.displayConfig();
        officePC.displayConfig();
    }
}
```

OUTPUT:



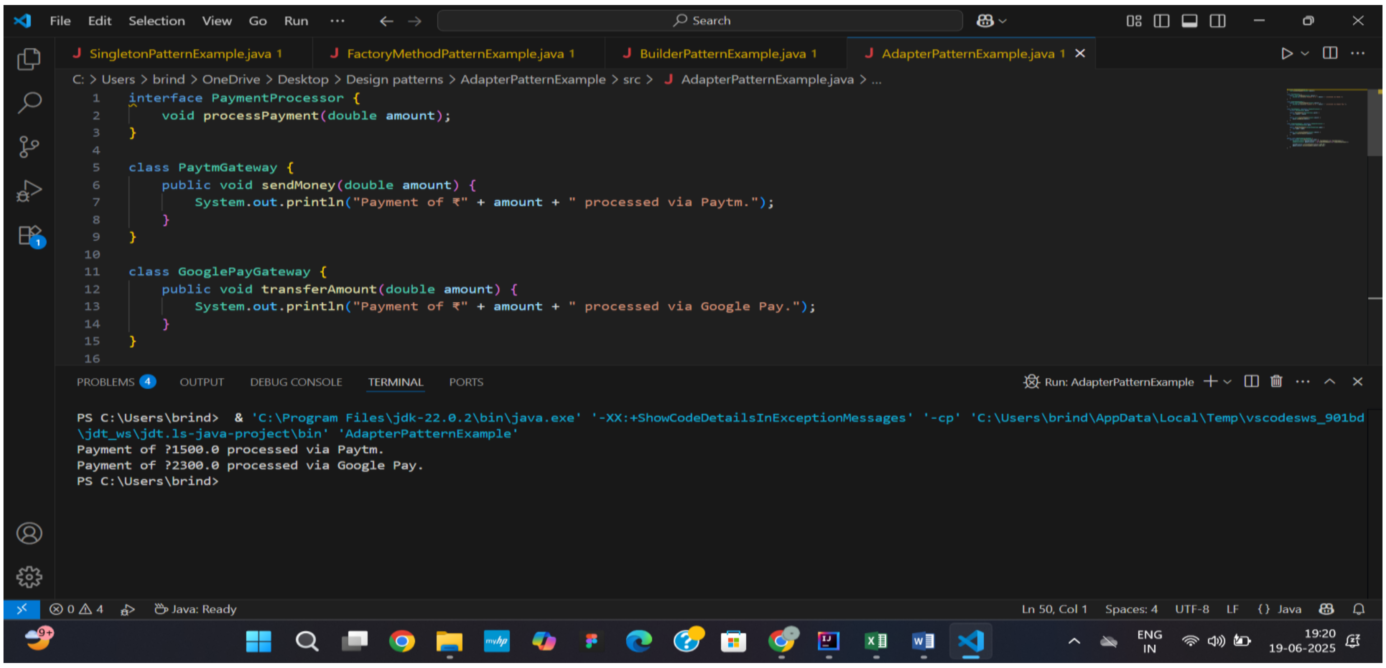
```
PS C:\Users\brind>  & 'C:\Program Files\jdk-22.0.2\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\brind\AppData\Local\Temp
\vscodesws_901bd\jdt_ws\jdt.ls-java-project\bin' 'BuilderPatternExample'
CPU: Intel i9
RAM: 32GB
Storage: 1TB SSD
GPU: NVIDIA RTX 4080
WiFi: true
Bluetooth: true
----------------------
CPU: Intel i5
RAM: 16GB
Storage: 512GB SSD
GPU: null
WiFi: true
Bluetooth: false
----------------------
PS C:\Users\brind>
```

Exercise 4: Implementing the Adapter Pattern

```java
interface PaymentProcessor {
    void processPayment(double amount);
}

class PaytmGateway {
    public void sendMoney(double amount) {
        System.out.println("Payment of ₹" + amount + " processed via Paytm.");
    }
}

class GooglePayGateway {
    public void transferAmount(double amount) {
        System.out.println("Payment of ₹" + amount + " processed via Google Pay.");
    }
}

class PaytmAdapter implements PaymentProcessor {
    private PaytmGateway paytm;

    public PaytmAdapter(PaytmGateway paytm) {
        this.paytm = paytm;
    }

    public void processPayment(double amount) {
        paytm.sendMoney(amount);
    }
}
```

```java
class GooglePayAdapter implements PaymentProcessor {
    private GooglePayGateway gpay;

    public GooglePayAdapter(GooglePayGateway gpay) {
        this.gpay = gpay;
    }

    public void processPayment(double amount) {
        gpay.transferAmount(amount);
    }
}

public class AdapterPatternExample {
    public static void main(String[] args) {
        PaymentProcessor paytmProcessor = new PaytmAdapter(new PaytmGateway());
        PaymentProcessor gpayProcessor = new GooglePayAdapter(new GooglePayGateway());

        paytmProcessor.processPayment(1500.00);
        gpayProcessor.processPayment(2300.00);
    }
}
```

OUTPUT:



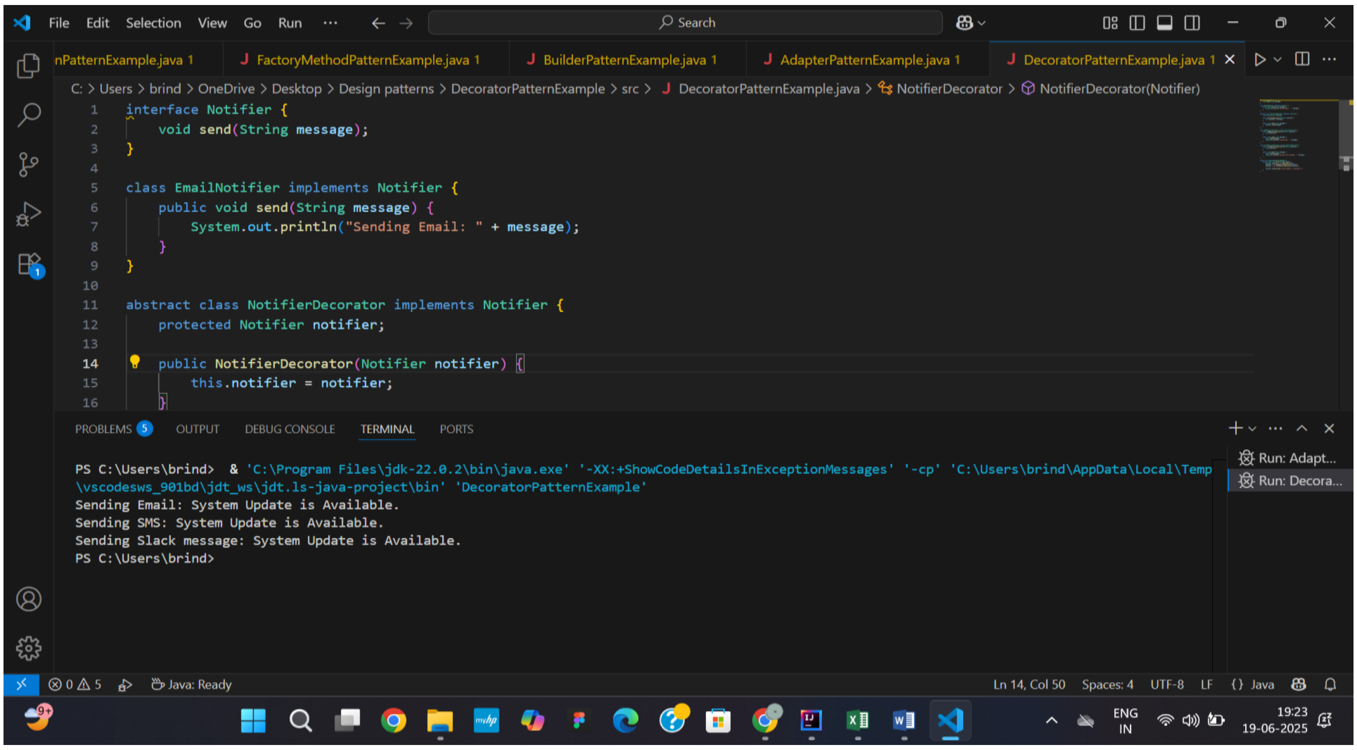
```java
interface PaymentProcessor {
    void processPayment(double amount);
}

class PaytmGateway {
    public void sendMoney(double amount) {
        System.out.println("Payment of ₹" + amount + " processed via Paytm.");
    }
}

class GooglePayGateway {
    public void transferAmount(double amount) {
        System.out.println("Payment of ₹" + amount + " processed via Google Pay.");
    }
}

```

```
PS C:\Users\brind>  & 'C:\Program Files\jdk-22.0.2\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\brind\AppData\Local\Temp\vscodesws_901bd
\jdt_ws\jdt.ls-java-project\bin' 'AdapterPatternExample'
Payment of ?1500.0 processed via Paytm.
Payment of ?2300.0 processed via Google Pay.
PS C:\Users\brind>
```

Exercise 5: Implementing the Decorator Pattern

```java
interface Notifier {
    void send(String message);
}

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

abstract class NotifierDecorator implements Notifier {
    protected Notifier notifier;

    public NotifierDecorator(Notifier notifier) {
        this.notifier = notifier;
    }

    public void send(String message) {
        notifier.send(message);
    }
}

class SMSNotifierDecorator extends NotifierDecorator {
    public SMSNotifierDecorator(Notifier notifier) {
        super(notifier);
    }

    public void send(String message) {
        super.send(message);
        System.out.println("Sending SMS: " + message);
    }
}

class SlackNotifierDecorator extends NotifierDecorator {
    public SlackNotifierDecorator(Notifier notifier) {
        super(notifier);
    }

    public void send(String message) {
        super.send(message);
        System.out.println("Sending Slack message: " + message);
    }
}

public class DecoratorPatternExample {
    public static void main(String[] args) {
        Notifier notifier = new EmailNotifier();
        notifier = new SMSNotifierDecorator(notifier);
        notifier = new SlackNotifierDecorator(notifier);

        notifier.send("System Update is Available.");
    }
}
```

OUTPUT:



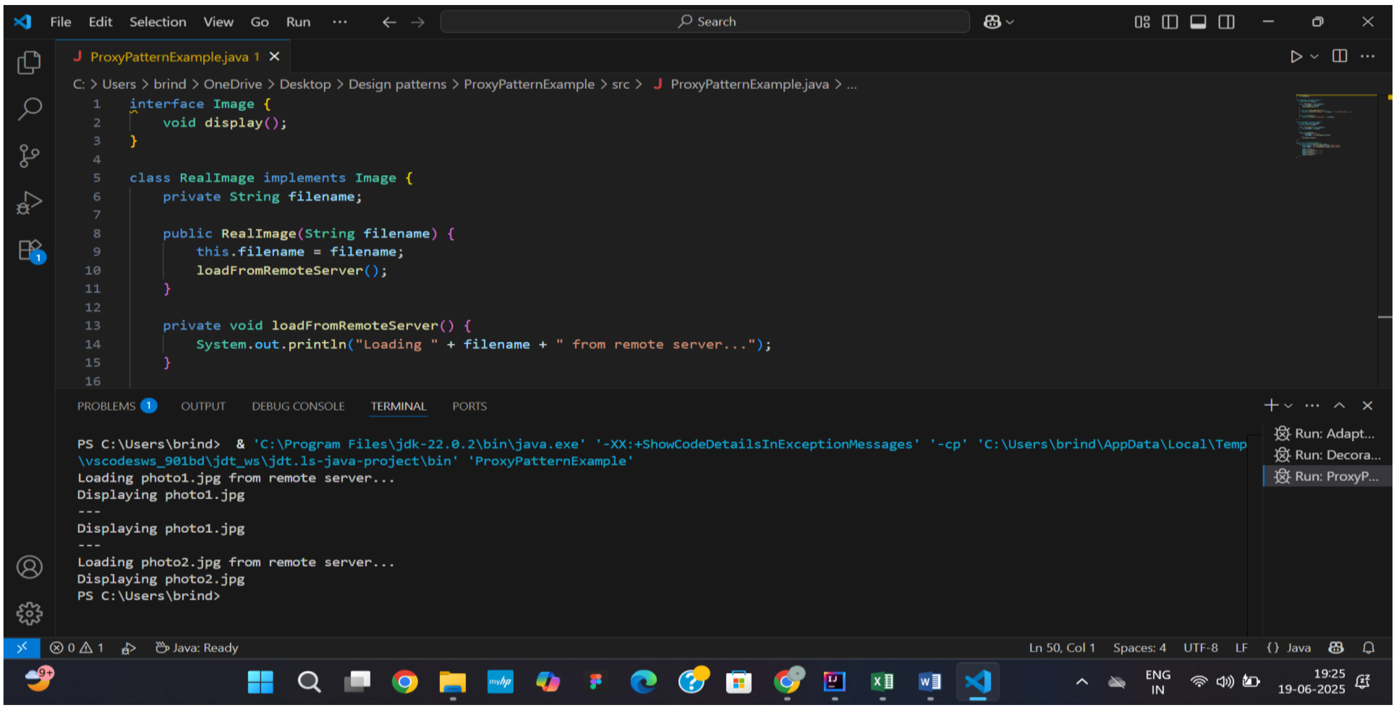
```java
interface Notifier {
    void send(String message);
}

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

abstract class NotifierDecorator implements Notifier {
    protected Notifier notifier;

    public NotifierDecorator(Notifier notifier) {
        this.notifier = notifier;
    }
```

```
PS C:\Users\brind>  & 'C:\Program Files\jdk-22.0.2\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\brind\AppData\Local\Temp
\vscodesws_901bd\jdt_ws\jdt.ls-java-project\bin' 'DecoratorPatternExample'
Sending Email: System Update is Available.
Sending SMS: System Update is Available.
Sending Slack message: System Update is Available.
PS C:\Users\brind>
```

Exercise 6: Implementing the Proxy Pattern

```java
interface Image {
    void display();
}

class RealImage implements Image {
    private String filename;

    public RealImage(String filename) {
        this.filename = filename;
        loadFromRemoteServer();
    }

    private void loadFromRemoteServer() {
        System.out.println("Loading " + filename + " from remote server...");
    }

    public void display() {
        System.out.println("Displaying " + filename);
    }
}

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 class ProxyPatternExample {
    public static void main(String[] args) {
        Image image1 = new ProxyImage("photo1.jpg");
        Image image2 = new ProxyImage("photo2.jpg");

        image1.display();
        System.out.println("---");
        image1.display();
        System.out.println("---");
        image2.display();
    }
}
```

OUTPUT:


File Edit Selection View Go Run
Search
ProxyPatternExample.java 1
C: > Users > brind > OneDrive > Desktop > Design patterns > ProxyPatternExample > src > ProxyPatternExample.java > ...
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Run: Adapt...
Run: Decora...
Run: ProxyP...
Java: Ready
Ln 50, Col 1
Spaces: 4
UTF-8
LF
{} Java
ENG IN
19:25
19-06-2025

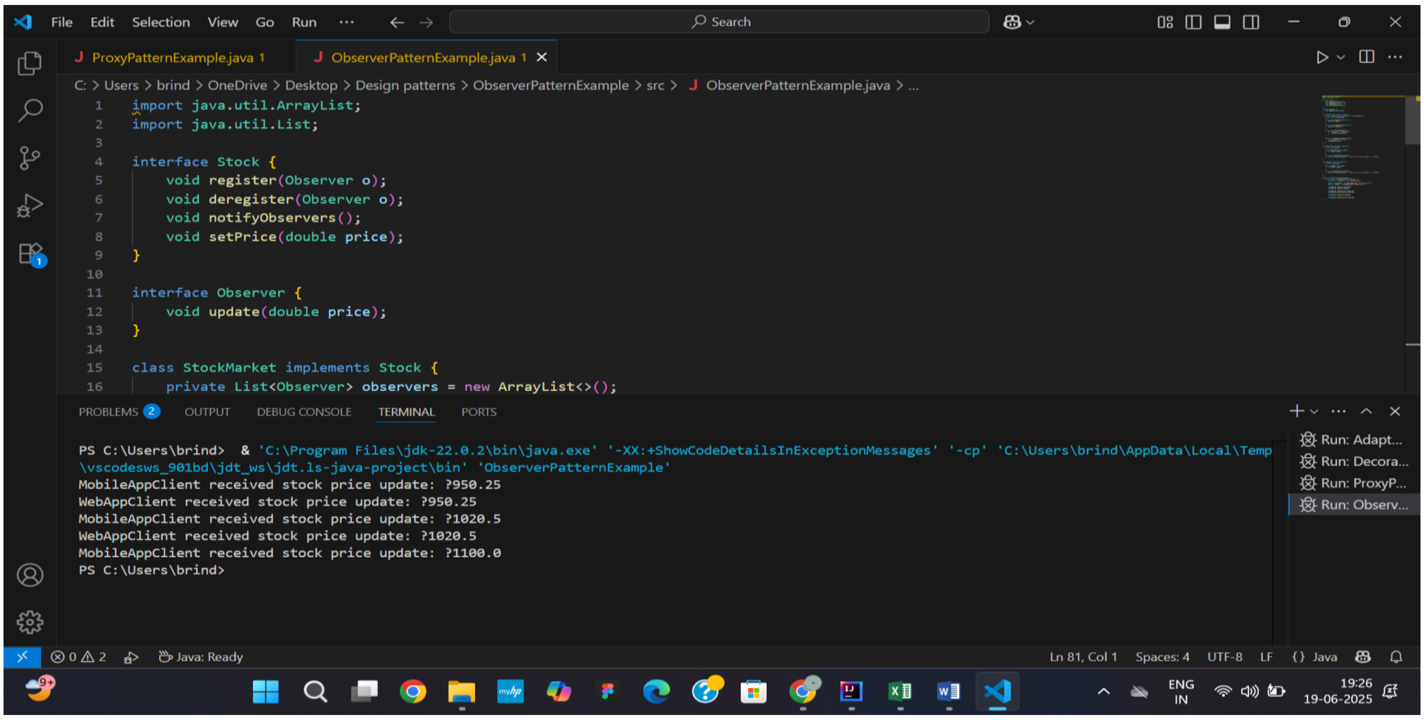
```java
interface Image {
    void display();
}

class RealImage implements Image {
    private String filename;

    public RealImage(String filename) {
        this.filename = filename;
        loadFromRemoteServer();
    }

    private void loadFromRemoteServer() {
        System.out.println("Loading " + filename + " from remote server...");
    }

```

```
PS C:\Users\brind>  & 'C:\Program Files\jdk-22.0.2\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\brind\AppData\Local\Temp
\vscodesws_901bd\jdt_ws\jdt.ls-java-project\bin' 'ProxyPatternExample'
Loading photo1.jpg from remote server...
Displaying photo1.jpg
---
Displaying photo1.jpg
---
Loading photo2.jpg from remote server...
Displaying photo2.jpg
PS C:\Users\brind>
```

Exercise 7: Implementing the Observer Pattern

```java
import java.util.ArrayList;
import java.util.List;

interface Stock {
    void register(Observer o);
    void deregister(Observer o);
    void notifyObservers();
    void setPrice(double price);
}

interface Observer {
    void update(double price);
}

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();
    }
}

class MobileApp implements Observer {
    private String name;

    public MobileApp(String name) {
        this.name = name;
    }

    public void update(double price) {
        System.out.println(name + " received stock price update: ₹" + price);
    }
}

class WebApp implements Observer {
    private String name;

    public WebApp(String name) {
        this.name = name;
    }

    public void update(double price) {
        System.out.println(name + " received stock price update: ₹" + price);
    }
}

public class ObserverPatternExample {
    public static void main(String[] args) {
        StockMarket stockMarket = new StockMarket();

        Observer mobileApp = new MobileApp("MobileAppClient");
        Observer webApp = new WebApp("WebAppClient");

        stockMarket.register(mobileApp);
        stockMarket.register(webApp);

        stockMarket.setPrice(950.25);
        stockMarket.setPrice(1020.50);

        stockMarket.deregister(webApp);

        stockMarket.setPrice(1100.00);
    }
}
```

OUTPUT:


File Edit Selection View Go Run ...
Search
ProxyPatternExample.java 1
ObserverPatternExample.java 1
C: > Users > brind > OneDrive > Desktop > Design patterns > ObserverPatternExample > src > ObserverPatternExample.java > ...
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Run: Adapt...
Run: Decora...
Run: ProxyP...
Run: Observ...
0 2
Java: Ready
Ln 81, Col 1
Spaces: 4
UTF-8
LF
{} Java
ENG IN
19:26
19-06-2025

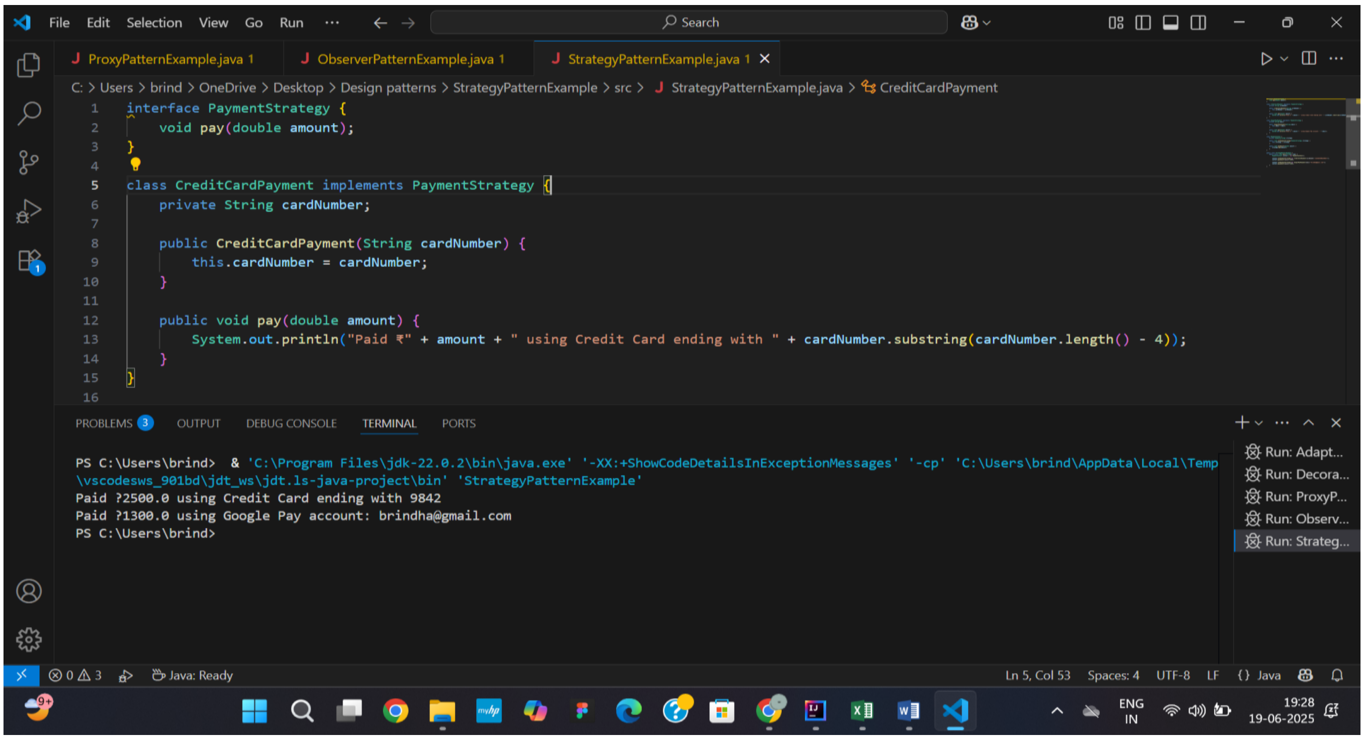
```java
import java.util.ArrayList;
import java.util.List;

interface Stock {
    void register(Observer o);
    void deregister(Observer o);
    void notifyObservers();
    void setPrice(double price);
}

interface Observer {
    void update(double price);
}

class StockMarket implements Stock {
    private List<Observer> observers = new ArrayList<>();
```

```
PS C:\Users\brind>  & 'C:\Program Files\jdk-22.0.2\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\brind\AppData\Local\Temp
\vscodesws_901bd\jdt_ws\jdt.ls-java-project\bin' 'ObserverPatternExample'
MobileAppClient received stock price update: ?950.25
WebAppClient received stock price update: ?950.25
MobileAppClient received stock price update: ?1020.5
WebAppClient received stock price update: ?1020.5
MobileAppClient received stock price update: ?1100.0
PS C:\Users\brind>
```

Exercise 8: Implementing the Strategy Pattern

```java
interface PaymentStrategy {
    void pay(double amount);
}

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 with " +
cardNumber.substring(cardNumber.length() - 4));
    }
}

class GooglePayPayment implements PaymentStrategy {
    private String email;

    public GooglePayPayment(String email) {
        this.email = email;
    }

    public void pay(double amount) {
        System.out.println("Paid ₹" + amount + " using Google Pay account: " + email);
    }
}

class PaymentContext {
    private PaymentStrategy strategy;

    public void setPaymentStrategy(PaymentStrategy strategy) {
        this.strategy = strategy;
    }

    public void payAmount(double amount) {
        strategy.pay(amount);
    }
}

public class StrategyPatternExample {
    public static void main(String[] args) {
        PaymentContext context = new PaymentContext();

        context.setPaymentStrategy(new CreditCardPayment("1234567890129842"));
        context.payAmount(2500);

        context.setPaymentStrategy(new GooglePayPayment("brindha@gmail.com"));
        context.payAmount(1300);
    }
}
```

OUTPUT:



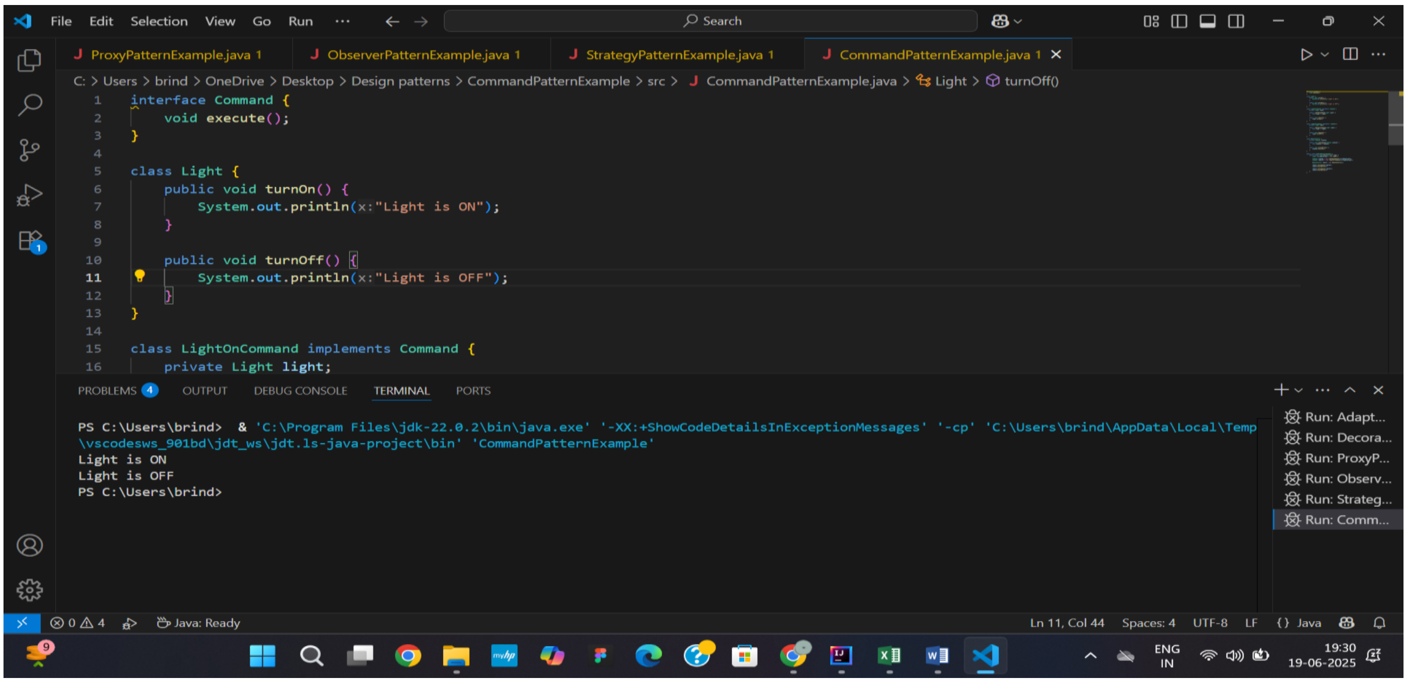
```java
interface PaymentStrategy {
    void pay(double amount);
}

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 with " + cardNumber.substring(cardNumber.length() - 4));
    }
}
```

```
PS C:\Users\brind>  & 'C:\Program Files\jdk-22.0.2\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\brind\AppData\Local\Temp
\vscodesws_901bd\jdt_ws\jdt.ls-java-project\bin' 'StrategyPatternExample'
Paid ?2500.0 using Credit Card ending with 9842
Paid ?1300.0 using Google Pay account: brindha@gmail.com
PS C:\Users\brind>
```

Exercise 9: Implementing the Command Pattern

```java
interface Command {
    void execute();
}

class Light {
    public void turnOn() {
        System.out.println("Light is ON");
    }

    public void turnOff() {
        System.out.println("Light is OFF");
    }
}

class LightOnCommand implements Command {
    private Light light;

    public LightOnCommand(Light light) {
        this.light = light;
    }

    public void execute() {
        light.turnOn();
    }
}

class LightOffCommand implements Command {
    private Light light;
    public LightOffCommand(Light light) {
        this.light = light;
    }

    public void execute() {
        light.turnOff();
    }
}

class RemoteControl {
    private Command command;

    public void setCommand(Command command) {
        this.command = command;
    }

    public void pressButton() {
        command.execute();
    }
}

public class CommandPatternExample {
    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:



```java
interface Command {
    void execute();
}

class Light {
    public void turnOn() {
        System.out.println(x:"Light is ON");
    }

    public void turnOff() {
        System.out.println(x:"Light is OFF");
    }
}

class LightOnCommand implements Command {
    private Light light;
```

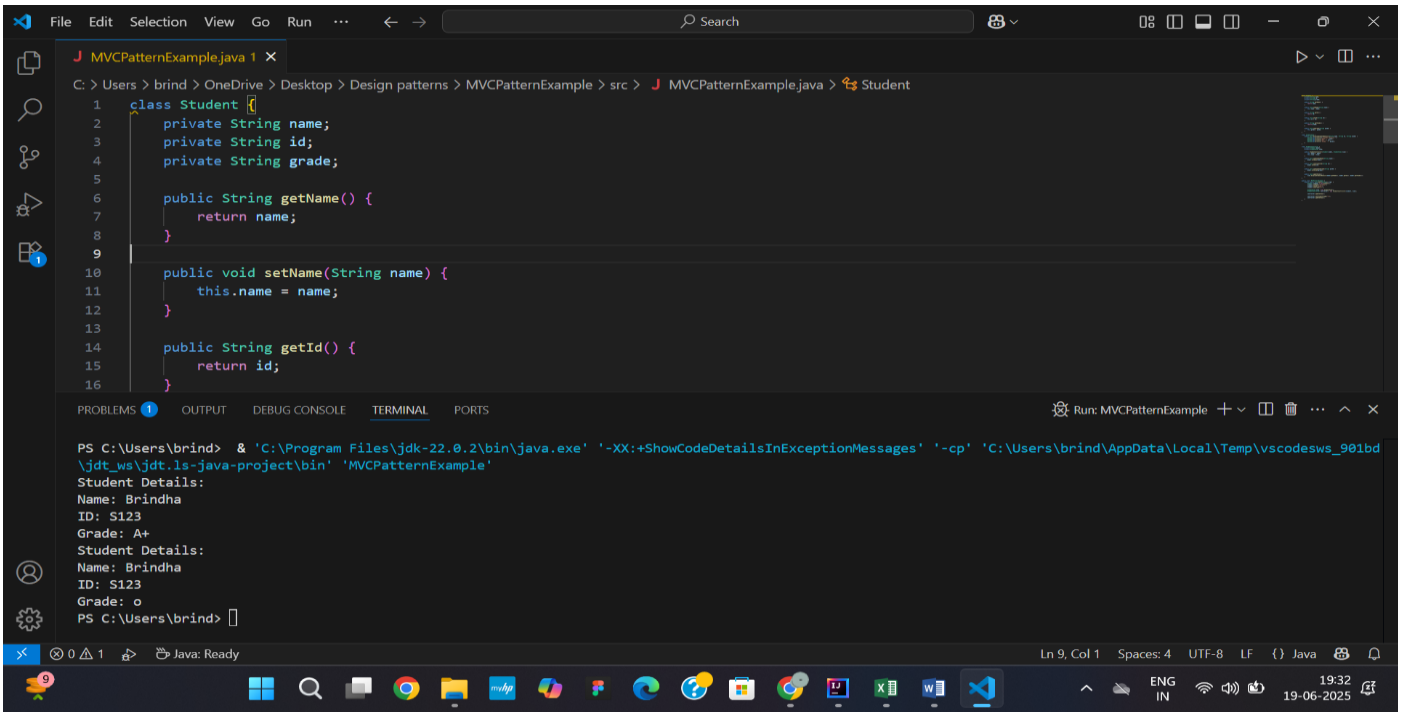
```
PS C:\Users\brind>  & 'C:\Program Files\jdk-22.0.2\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\brind\AppData\Local\Temp
\vscodesws_901bd\jdt_ws\jdt.ls-java-project\bin' 'CommandPatternExample'
Light is ON
Light is OFF
PS C:\Users\brind>
```

Exercise 10: Implementing the MVC Pattern

```java
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;
    }
}
```

```java
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);
    }
}

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 MVCPatternExample {
    public static void main(String[] args) {
        Student student = new Student();
        student.setName("Brindha");
        student.setId("S123");
        student.setGrade("A+");

        StudentView view = new StudentView();
        StudentController controller = new StudentController(student, view);

        controller.updateView();

        controller.setStudentGrade("o");
        controller.updateView();
    }
}
```

OUTPUT:



```
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;
    }
```

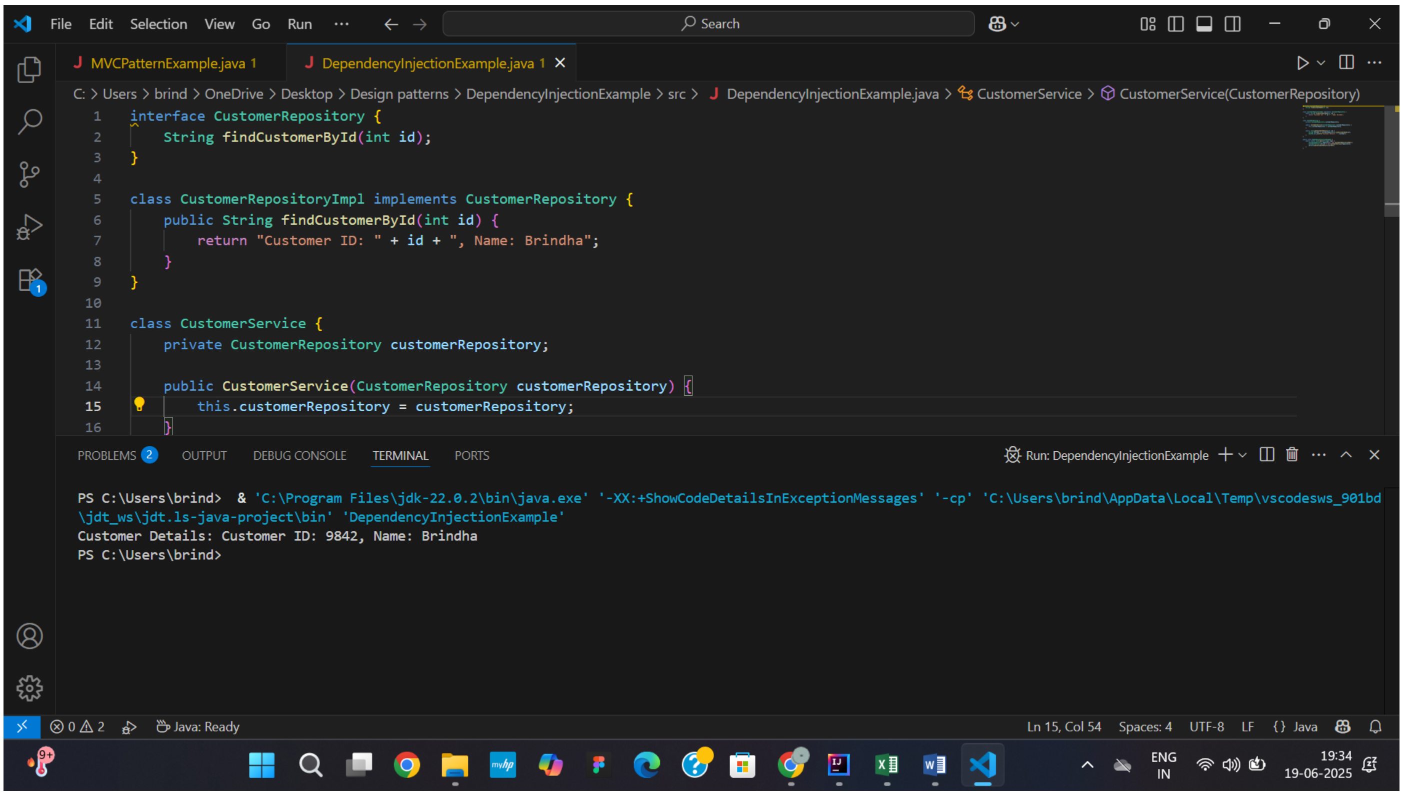
```
PS C:\Users\brind>  & 'C:\Program Files\jdk-22.0.2\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\brind\AppData\Local\Temp\vscodesws_901bd
\jdt_ws\jdt.ls-java-project\bin' 'MVCPatternExample'
Student Details:
Name: Brindha
ID: S123
Grade: A+
Student Details:
Name: Brindha
ID: S123
Grade: o
PS C:\Users\brind> 
```

File Edit Selection View Go Run
Search
MVCPatternExample.java 1
C: > Users > brind > OneDrive > Desktop > Design patterns > MVCPatternExample > src > MVCPatternExample.java > Student
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Run: MVCPatternExample
0 1 Java: Ready
Ln 9, Col 1 Spaces: 4 UTF-8 LF {} Java
ENG IN
19:32
19-06-2025

Exercise 11: Implementing Dependency Injection

```java
interface CustomerRepository {
    String findCustomerById(int id);
}

class CustomerRepositoryImpl implements CustomerRepository {
    public String findCustomerById(int id) {
        return "Customer ID: " + id + ", Name: Brindha";
    }
}

class CustomerService {
    private CustomerRepository customerRepository;

    public CustomerService(CustomerRepository customerRepository) {
        this.customerRepository = customerRepository;
    }

    public void getCustomerDetails(int id) {
        String customer = customerRepository.findCustomerById(id);
        System.out.println("Customer Details: " + customer);
    }
}

public class DependencyInjectionExample {
    public static void main(String[] args) {
        CustomerRepository repository = new CustomerRepositoryImpl();
        CustomerService service = new CustomerService(repository);
        service.getCustomerDetails(9842);
    }
}
```

OUTPUT:


File Edit Selection View Go Run ···
Search
MVCPatternExample.java 1
DependencyInjectionExample.java 1
C: > Users > brind > OneDrive > Desktop > Design patterns > DependencyInjectionExample > src > DependencyInjectionExample.java > CustomerService > CustomerService(CustomerRepository)
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Run: DependencyInjectionExample
Java: Ready
Ln 15, Col 54
Spaces: 4
UTF-8
LF
{} Java
ENG IN
19:34
19-06-2025

```java
interface CustomerRepository {
    String findCustomerById(int id);
}

class CustomerRepositoryImpl implements CustomerRepository {
    public String findCustomerById(int id) {
        return "Customer ID: " + id + ", Name: Brindha";
    }
}

class CustomerService {
    private CustomerRepository customerRepository;

    public CustomerService(CustomerRepository customerRepository) {
        this.customerRepository = customerRepository;
    }
```

```
PS C:\Users\brind>  & 'C:\Program Files\jdk-22.0.2\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\brind\AppData\Local\Temp\vscodesws_901bd
\jdt_ws\jdt.ls-java-project\bin' 'DependencyInjectionExample'
Customer Details: Customer ID: 9842, Name: Brindha
PS C:\Users\brind>
```