**Exercise 1: Implementing the Singleton Pattern**

**CODE:**

**Logger.java**

package SingletonPatternExample;

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**

package SingletonPatternExample;

public class Main {

    public static void main(String[] args) {

        // Simulate logger in application layer

        Logger appLogger = Logger.getInstance();

        appLogger.log("Application started.");

// Simulate logger in database layer

        Logger dbLogger = Logger.getInstance();

        dbLogger.log("Connected to database.");

        // Check if both references point to the same instance

        if (appLogger == dbLogger) {

            System.out.println("Confirmed: Only one Logger instance exists.");

        } else {

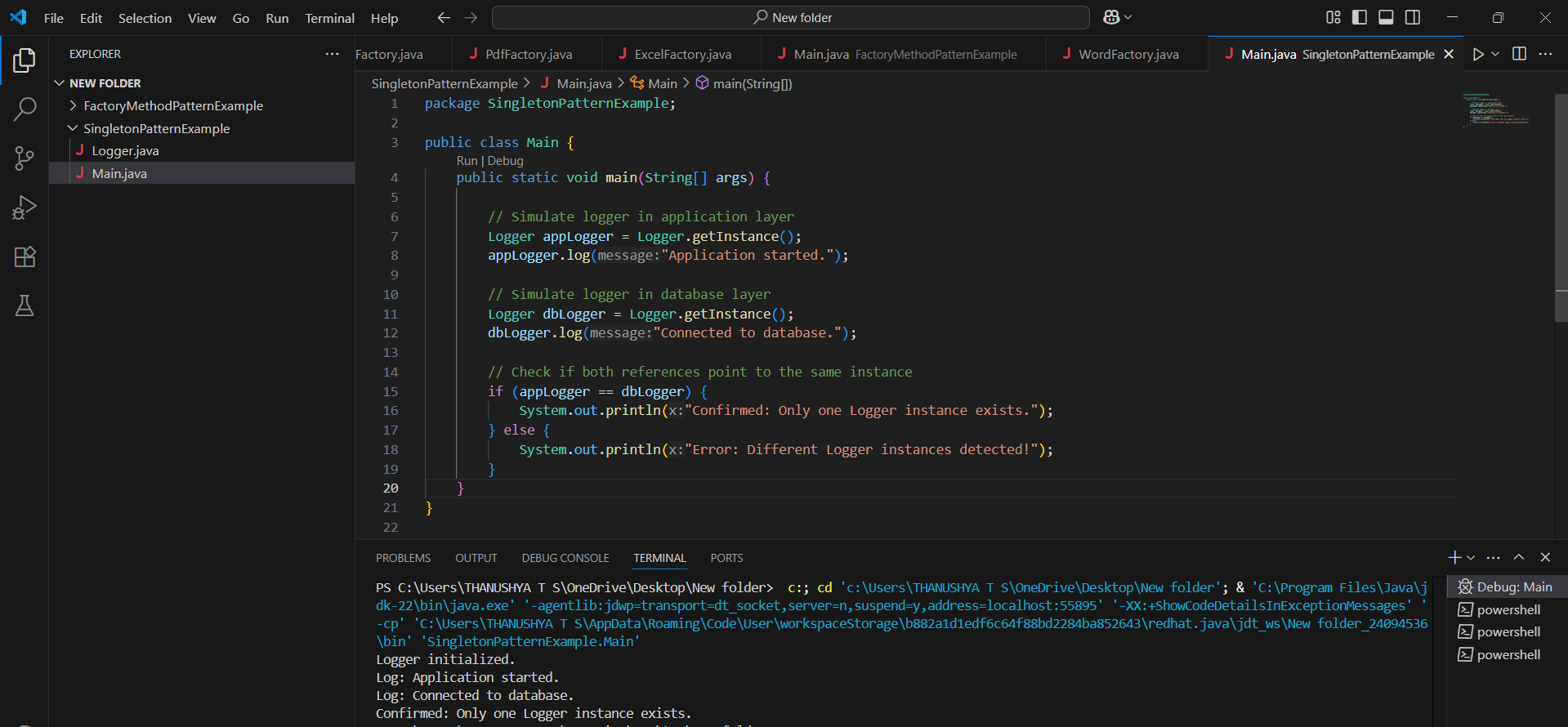
            System.out.println("Error: Different Logger instances detected!");

        }

    }

}

**OUTPUT:**

****

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

**CODE:**

**Document.java**

package FactoryMethodPatternExample;

public interface Document {

    void open();

}

**Word.java**

package FactoryMethodPatternExample;

public class Word implements Document {

    @Override

    public void open() {

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

    }

}

**Pdf.java**

package FactoryMethodPatternExample;

public class Pdf implements Document {

    @Override

    public void open() {

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

    }

}

**Excel.java**

package FactoryMethodPatternExample;

public class Excel implements Document {

    @Override

    public void open() {

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

    }

}

**DocumentFactory.java**

package FactoryMethodPatternExample;

public abstract class DocumentFactory {

    public abstract Document createDocument();

}

**WordFactory.java**

package FactoryMethodPatternExample;

public class WordFactory extends DocumentFactory {

    @Override

    public Document createDocument() {

        return new Word();

    }

}

**PdfFactory.java**

package FactoryMethodPatternExample;

public class PdfFactory extends DocumentFactory {

    @Override

    public Document createDocument() {

        return new Pdf();

    }

}

**ExcelFactory.java**

package FactoryMethodPatternExample;

public class ExcelFactory extends DocumentFactory {

    @Override

    public Document createDocument() {

        return new Excel();

    }

}

**Main.java**

package FactoryMethodPatternExample;

public class Main {

    public static void main(String[] args) {

        DocumentFactory wordFactory = new WordFactory();

        Document wordDoc = wordFactory.createDocument();

        wordDoc.open();

        DocumentFactory pdfFactory = new PdfFactory();

        Document pdfDoc = pdfFactory.createDocument();

        pdfDoc.open();

        DocumentFactory excelFactory = new ExcelFactory();

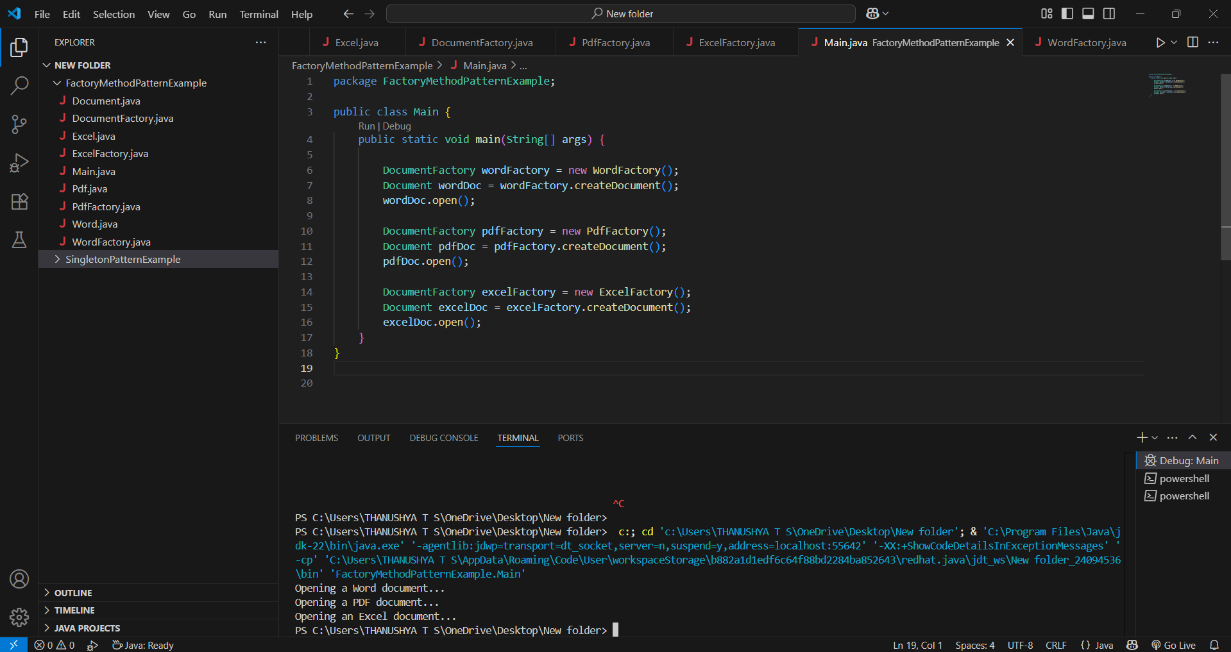
        Document excelDoc = excelFactory.createDocument();

        excelDoc.open();

    }

}

**OUTPUT:**



**Exercise 3: Implementing the Builder Pattern**

**CODE:**

**Computer.java**

package BuilderPatternExample;

public class Computer {

     private String CPU;

    private String RAM;

    private String storage;

    private String graphicsCard;

    private String operatingSystem;

    private Computer(Builder builder) {

        this.CPU = builder.CPU;

        this.RAM = builder.RAM;

        this.storage = builder.storage;

        this.graphicsCard = builder.graphicsCard;

        this.operatingSystem = builder.operatingSystem;

    }

    public static class Builder {

        private String CPU;

private String RAM;

        private String storage;

        private String graphicsCard;

        private String operatingSystem;

        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 setOperatingSystem(String operatingSystem) {

            this.operatingSystem = operatingSystem;

            return this;

        }

        public Computer build() {

            return new Computer(this);

        }

    }

    public void showSpecs() {

        System.out.println("Computer Configuration:");

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

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

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

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

        System.out.println("Operating System: " + (operatingSystem != null ? operatingSystem : "Not specified"));

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

    }

}

**Main.java**

package BuilderPatternExample;

public class Main {

     public static void main(String[] args) {

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

                .build();

        basicComputer.showSpecs();

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

                .setStorage("1TB SSD")

                .setGraphicsCard("NVIDIA RTX 4080")

                .setOperatingSystem("Windows 11")

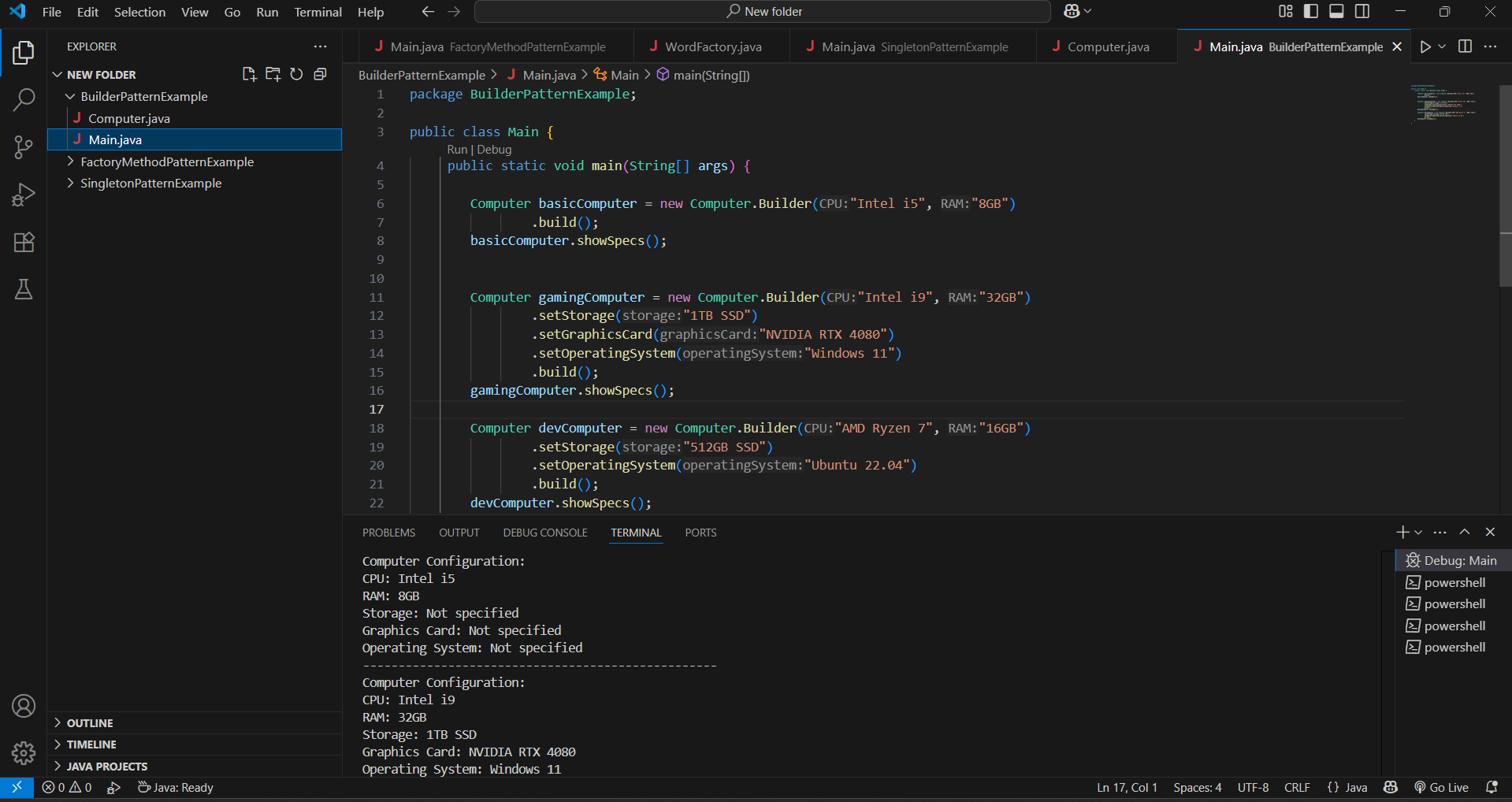
                .build();

        gamingComputer.showSpecs();

    }

}

**OUTPUT:**

****

**Exercise 6: Implementing the Proxy Pattern**

**CODE:**

**Image.java**

package ProxyPattern;

public interface Image {

    void display();

}

**RealImage.java**

package ProxyPattern;

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);

        // Simulate delay or network latency

        try {

            Thread.sleep(1000);  // Optional for realism

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

    }

    @Override

    public void display() {

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

    }

}

**ProxyImage.java**

package ProxyPattern;

public class ProxyImage implements Image {

    private RealImage realImage;

    private String fileName;

    public ProxyImage(String fileName) {

        this.fileName = fileName;

    }

    @Override

    public void display() {

        if (realImage == null) {

            realImage = new RealImage(fileName);  // Lazy initialization

        } else {

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

        }

        realImage.display();

    }

}

**Main.java**

package ProxyPattern;

public class Main {

    public static void main(String[] args) {

        Image img1 = new ProxyImage("landscape.jpg");

        Image img2 = new ProxyImage("portrait.jpg");

        img1.display();

        img1.display();

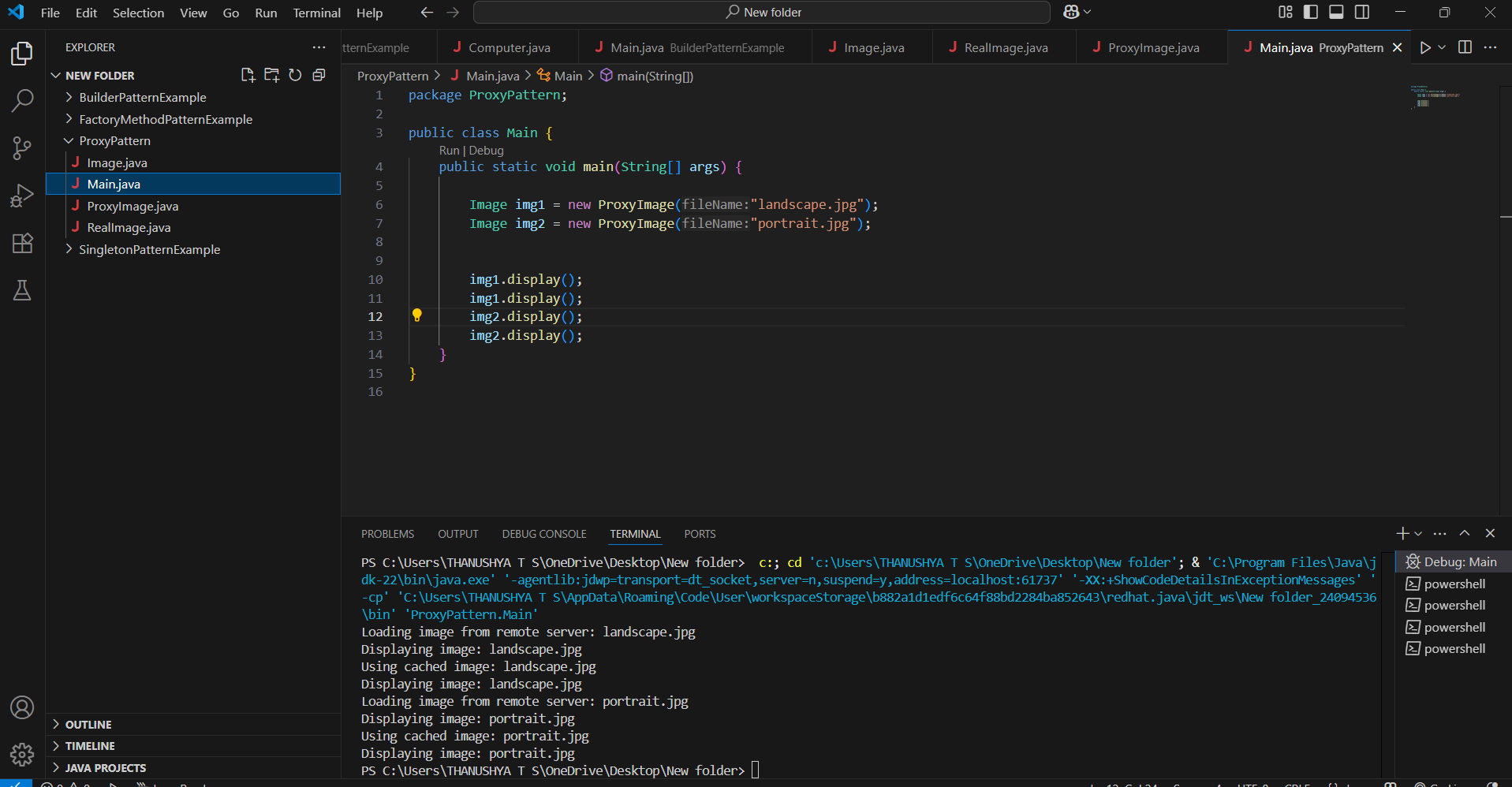
        img2.display();

        img2.display();

    }

}

**OUTPUT:**



**Exercise 10: Implementing the MVC Pattern**

**CODE:**

**Student.java**

package MVC;

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; }

}

**StudentView.java**

package MVC;

public class StudentView {

     public void displayStudentDetails(String name, String id, String grade) {

        System.out.println("\n Student Details");

        System.out.println("Name  : " + name);

        System.out.println("ID    : " + id);

        System.out.println("Grade : " + grade);

    }

}

**StudentController.java**

package MVC;

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 setStudentGrade(String grade) {

        model.setGrade(grade);

    }

    public String getStudentName() {

        return model.getName();

    }

    public String getStudentId() {

        return model.getId();

    }

    public String getStudentGrade() {

        return model.getGrade();

    }

    public void updateView() {

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

    }

}

**MVCDemo.java**

package MVC;

public class MVCDemo {

     public static void main(String[] args) {

        Student student = new Student("Alice", "S101", "A");

        StudentView view = new StudentView();

StudentController controller = new StudentController(student, view);

        controller.updateView();

        controller.setStudentName("Smith");

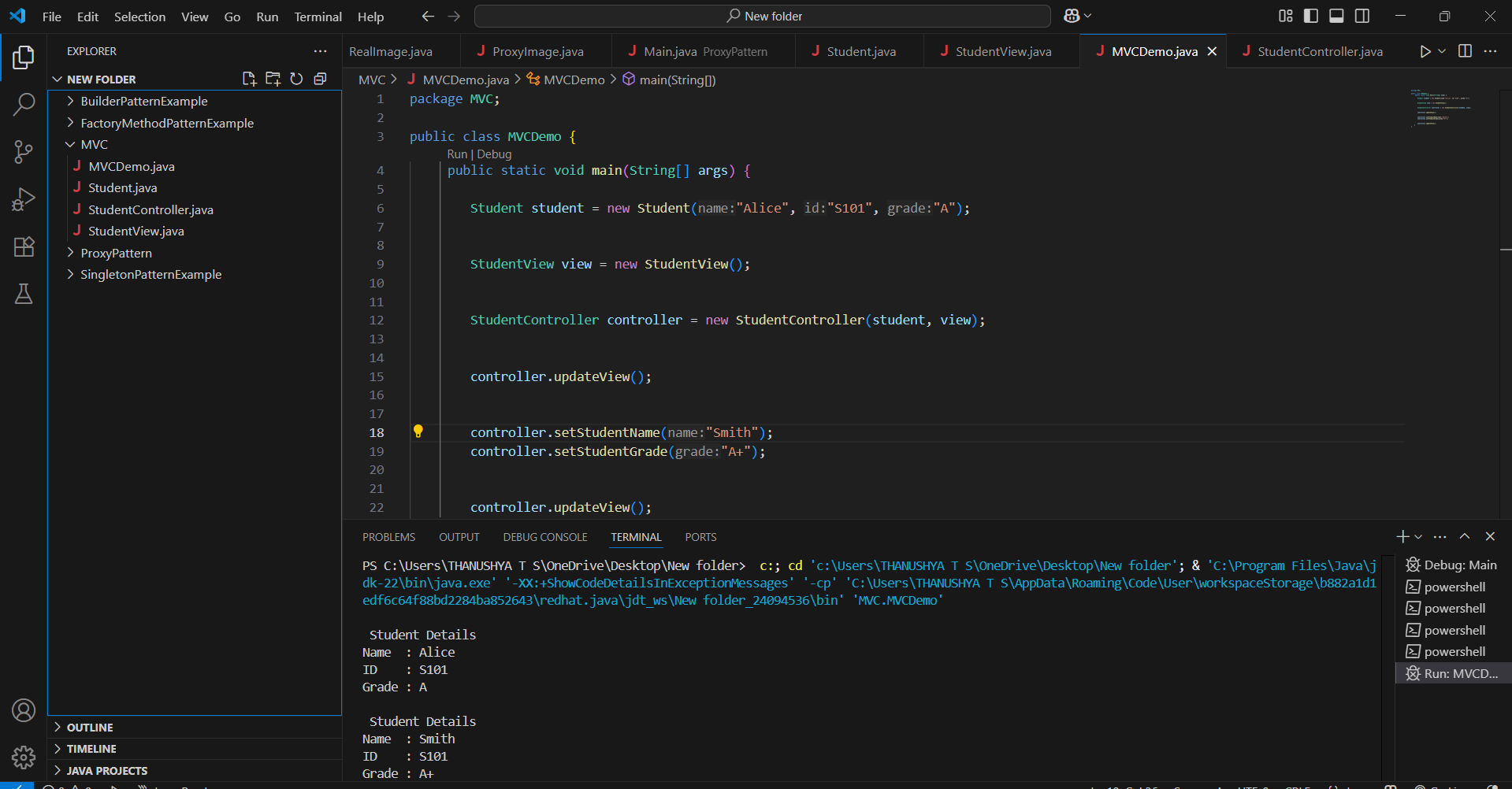
        controller.setStudentGrade("A+");

        controller.updateView();

    }

}

**OUTPUT:**

****

**Exercise 11: Implementing Dependency Injection**

**CODE:**

**CustomerRepository.java**

package DI;

public interface CustomerRepository {

    String findCustomerById(int id);

}

**CustomerRepositoryImpl.java**

package DI;

import java.util.HashMap;

import java.util.Map;

public class CustomerRepositoryImpl implements CustomerRepository {

    private Map<Integer, String> customers;

    public CustomerRepositoryImpl() {

        customers = new HashMap<>();

        customers.put(101, "Alice Johnson");

        customers.put(102, "Bob Smith");

        customers.put(103, "Charlie Rose");

    }

    @Override

    public String findCustomerById(int id) {

        return customers.getOrDefault(id, "Customer Not Found");

    }

}

**CustomerService.java**

package DI;

public class CustomerService {

    private final CustomerRepository customerRepository;

    public CustomerService(CustomerRepository customerRepository) {

        this.customerRepository = customerRepository;

    }

    public void getCustomerDetails(int customerId) {

        String customer = customerRepository.findCustomerById(customerId);

        System.out.println("🔍 Customer ID " + customerId + ": " + customer);

    }

}

**Main.java**

package DI;

public class Main {

       public static void main(String[] args) {

        CustomerRepository repo = new CustomerRepositoryImpl();

        CustomerService service = new CustomerService(repo);

        service.getCustomerDetails(101);

        service.getCustomerDetails(104);

    }

}

**OUTPUT:**

