**DesignPatterns&Principles**

**Exercise 1: Implementing the Singleton Pattern**

**Logger.java**

public class Logger {

    private static Logger instance;

    private Logger() {

    }

    public static Logger getInstance() {

        if (instance == null) {

            instance = new Logger();

        }

        return instance;

    }

    public void displayMessage() {

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

    }

}

**Main.java**

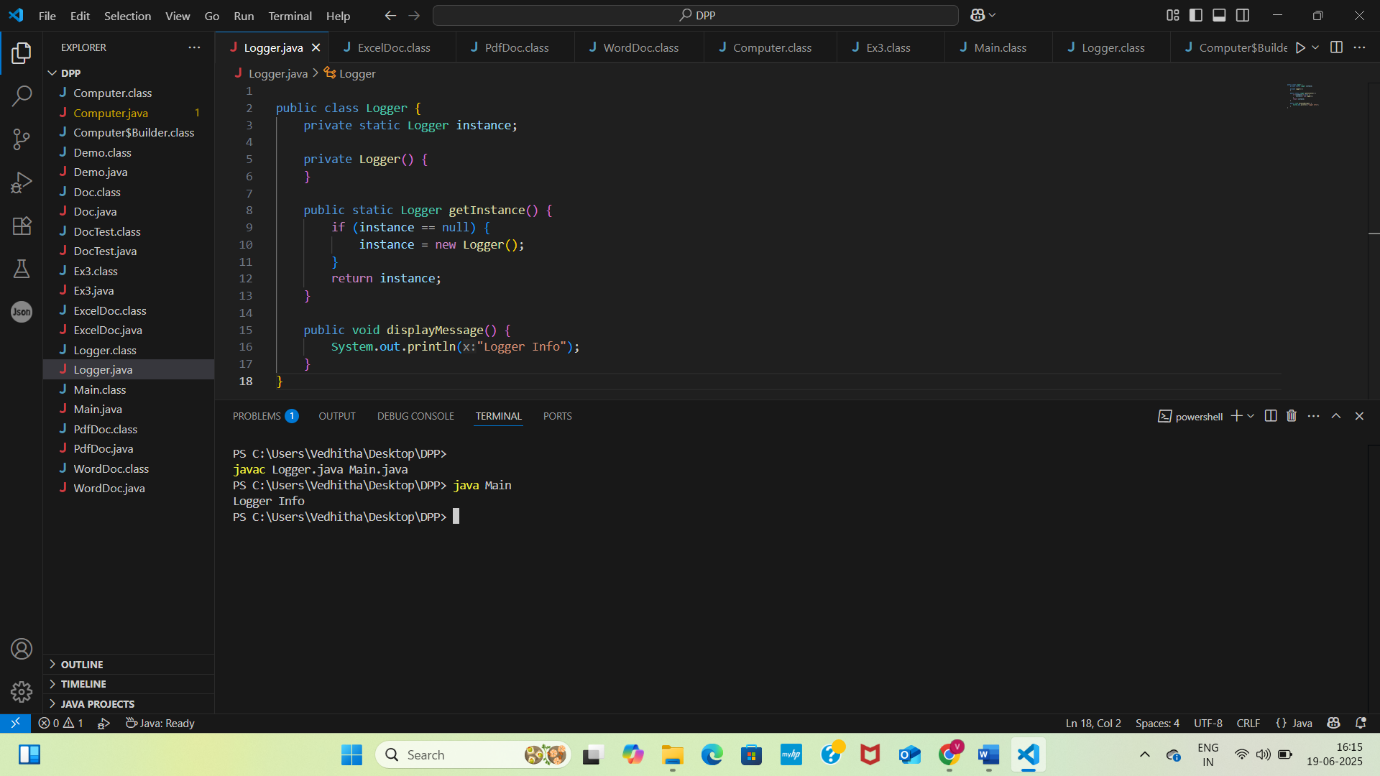
public class Main {

    public static void main(String args[]) {

        Logger.getInstance().displayMessage();

    }

}



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

**Doc.java**

public interface Doc {

    public void open();

}

**WordDoc.java**

public class WordDoc implements Doc {

    public void open() {

        System.out.println("WordDocument");

    }

}

**PdfDoc.java**

public class PdfDoc implements Doc {

    public void open() {

        System.out.println("PdfDocument");

    }

}

**ExcelDoc.java**

public class ExcelDoc implements Doc {

    public void open() {

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

    }

}

**DocTest.java**

public class DocTest {

    public static void main(String args[]) {

        Demo doc = new Demo();

        Doc word = doc.getDoc("WordDoc");

        if (word != null) {

            word.open();

        } else {

            System.out.println("Invalid");

        }

    }

}

**Demo.java**

public class Demo {

    public Doc getDoc(String docType) {

        if (docType == null) {

            return null;

        }

        if (docType.equalsIgnoreCase("WordDoc")) {

            return new WordDoc();

        } else if (docType.equalsIgnoreCase("PdfDoc")) {

            return new PdfDoc();

        } else if (docType.equalsIgnoreCase("ExcelDoc")) {

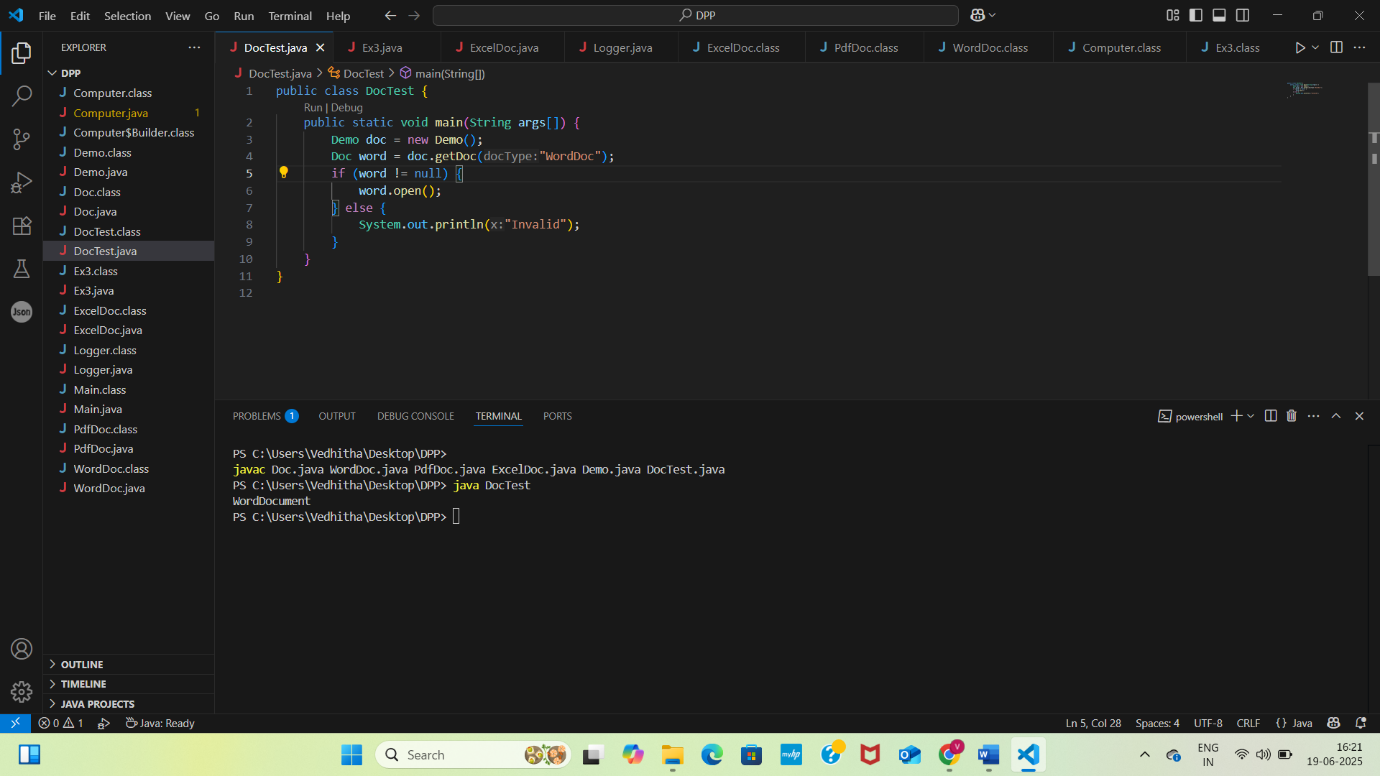
            return new ExcelDoc();

        }

        return null;

    }

}

****

**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 motherboard;

    private Computer(Builder builder) {

        this.CPU = builder.CPU;

        this.RAM = builder.RAM;

        this.storage = builder.storage;

        this.graphicsCard = builder.graphicsCard;

        this.motherboard = builder.motherboard;

    }

    public static class Builder {

        private String CPU;

        private String RAM;

        private String storage;

        private String graphicsCard;

        private String motherboard;

        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 setMotherboard(String motherboard) {

            this.motherboard = motherboard;

            return this;

        }

        public Computer build() {

            return new Computer(this);

        }

    }

    public String toString() {

        return "Computer [CPU=" + CPU + ", RAM=" + RAM + ", Storage=" + storage +

                ", GraphicsCard=" + graphicsCard + ", Motherboard=" + motherboard + "]";

    }

}

**Ex3.java**

public class Ex3 {

    public static void main(String[] args) {

        Computer basicComputer = new Computer.Builder("Intel i3", "4GB").build();

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

                .setStorage("2TB SSD")

                .setGraphicsCard("NVIDIA RTX 4090")

                .setMotherboard("Asus ROG Maximus")

                .build();

        Computer officeComputer = new Computer.Builder("AMD Ryzen 5", "16GB")

                .setStorage("512GB SSD")

                .build();

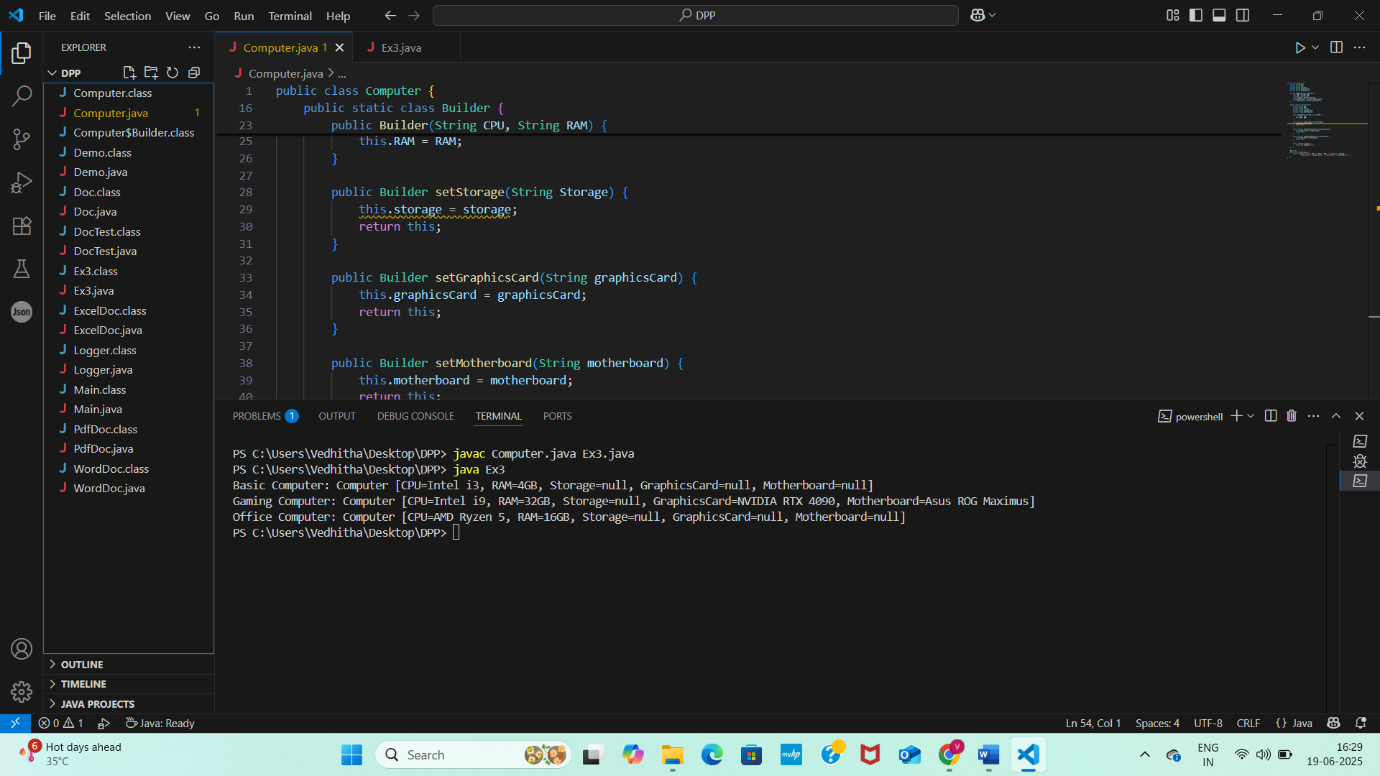
        System.out.println("Basic Computer: " + basicComputer);

        System.out.println("Gaming Computer: " + gamingComputer);

        System.out.println("Office Computer: " + officeComputer);

    }

}



**Exercise 4: Implementing the Adapter Pattern**

PaymentProceesor.java

**package** src;

**public** **interface** PaymentProceesor {

**public** **void** processPayment(**double** amount);

}

Paypal.java

**package** src;

**public** **class** Paypal {

**public** **void** makepayment(**double** amount) {

System.***out***.println(amount);

}

}

PaypalAdapter.java

**package** src;

**public** **class** PaypalAdapter **implements** PaymentProceesor{

**private** Paypal paypal;

**public** PaypalAdapter(Paypal paypal) {

**this**.paypal=paypal;

}

**public** **void** processPayment(**double** amount) {

paypal.makepayment(amount);

}

}

Stripe.java

**package** src;

**public** **class** PaypalAdapter **implements** PaymentProceesor{

**private** Paypal paypal;

**public** PaypalAdapter(Paypal paypal) {

**this**.paypal=paypal;

}

**public** **void** processPayment(**double** amount) {

paypal.makepayment(amount);

}

}

StripeAdapter.java

**package** src;

**public** **class** StripeAdapter **implements** PaymentProceesor{

**private** Stripe stripe;

**public** StripeAdapter(Stripe stripe) {

**this**.stripe=stripe;

}

**public** **void** processPayment(**double** amount) {

stripe.Stripepayment(amount);

}

}

Main.java

**package** src;

**public** **class** Main {

**public** **static** **void** main(String args[]) {

Paypal paypal=**new** Paypal();

PaymentProceesor paypalProceesor=**new** PaypalAdapter(paypal);

paypalProceesor.processPayment(15000.0);

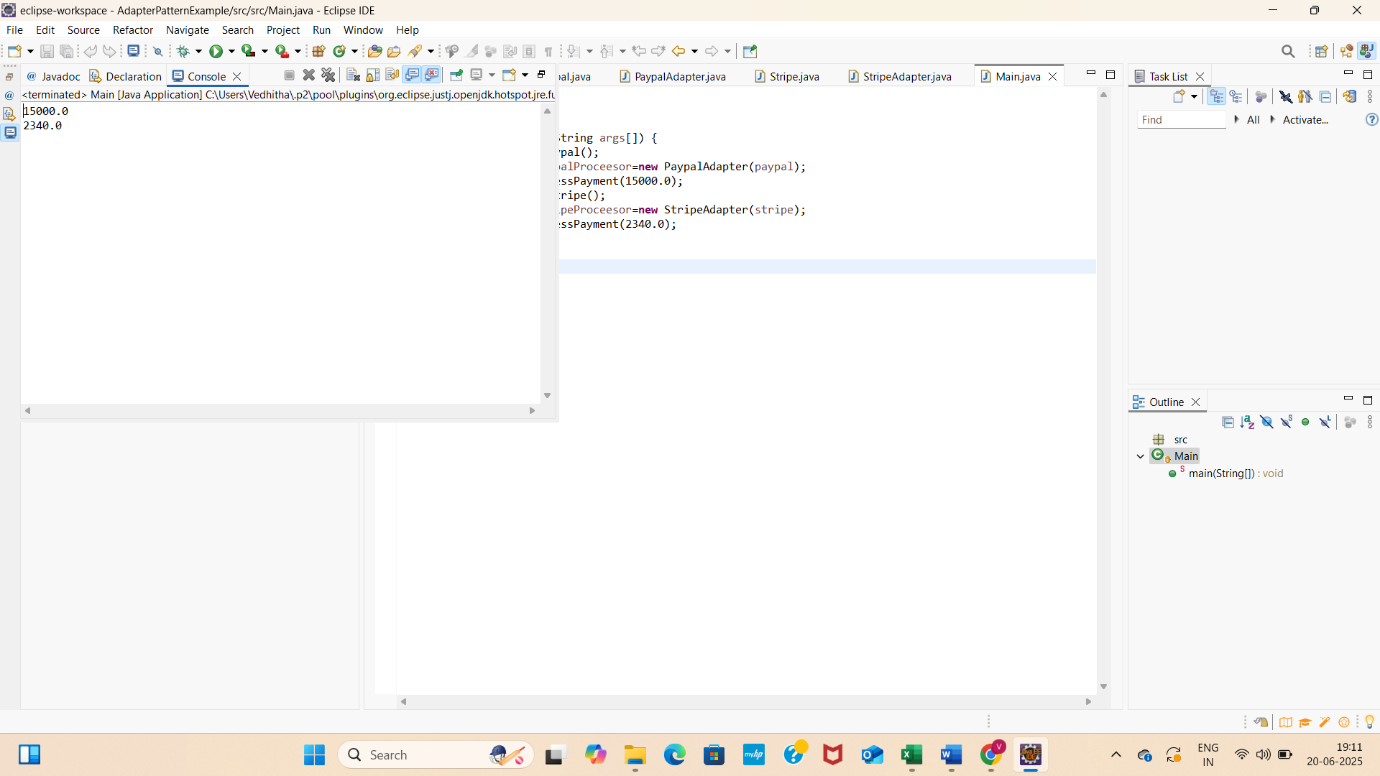
Stripe stripe =**new** Stripe();

PaymentProceesor stripeProceesor=**new** StripeAdapter(stripe);

stripeProceesor.processPayment(2340.0);

}

}



**Algorithms\_DataStructures**

**Exercise 2: E-commerce Platform Search Function**

Product.java

**package** src;

**public** **class** Product {

**int** productId;

String productName;

String category;

**public** Product(**int** productId, String productName, String category) {

**this**.productId = productId;

**this**.productName = productName;

**this**.category = category;

}

**public** String toString() {

**return** "[" + productId + ", " + productName + ", " + category + "]";

}

}

ProductSearch.java

**package** src;

**import** java.util.Arrays;

**import** java.util.Comparator;

**public** **class** ProductSearch {

**public** **static** Product linearSearch(Product[] products, **int** targetId) {

**for** (Product product : products) {

**if** (product.productId == targetId) {

**return** product;

}

}

**return** **null**;

}

**public** **static** Product binarySearch(Product[] products, **int** targetId) {

**int** low = 0;

**int** high = products.length - 1;

**while** (low <= high) {

**int** mid = (low + high) / 2;

**if** (products[mid].productId == targetId) {

**return** products[mid];

} **else** **if** (products[mid].productId < targetId) {

low = mid + 1;

} **else** {

high = mid - 1;

}

}

**return** **null**;

}

**public** **static** **void** main(String[] args) {

Product[] products = {

**new** Product(103, "Laptop", "Electronics"),

**new** Product(101, "Shoes", "Fashion"),

**new** Product(104, "Phone", "Electronics"),

**new** Product(102, "Book", "Education")

};

System.***out***.println("Linear Search:");

Product result1 = *linearSearch*(products, 104);

System.***out***.println(result1 != **null** ? result1 : "Product not found");

Arrays.*sort*(products, Comparator.*comparingInt*(p -> p.productId));

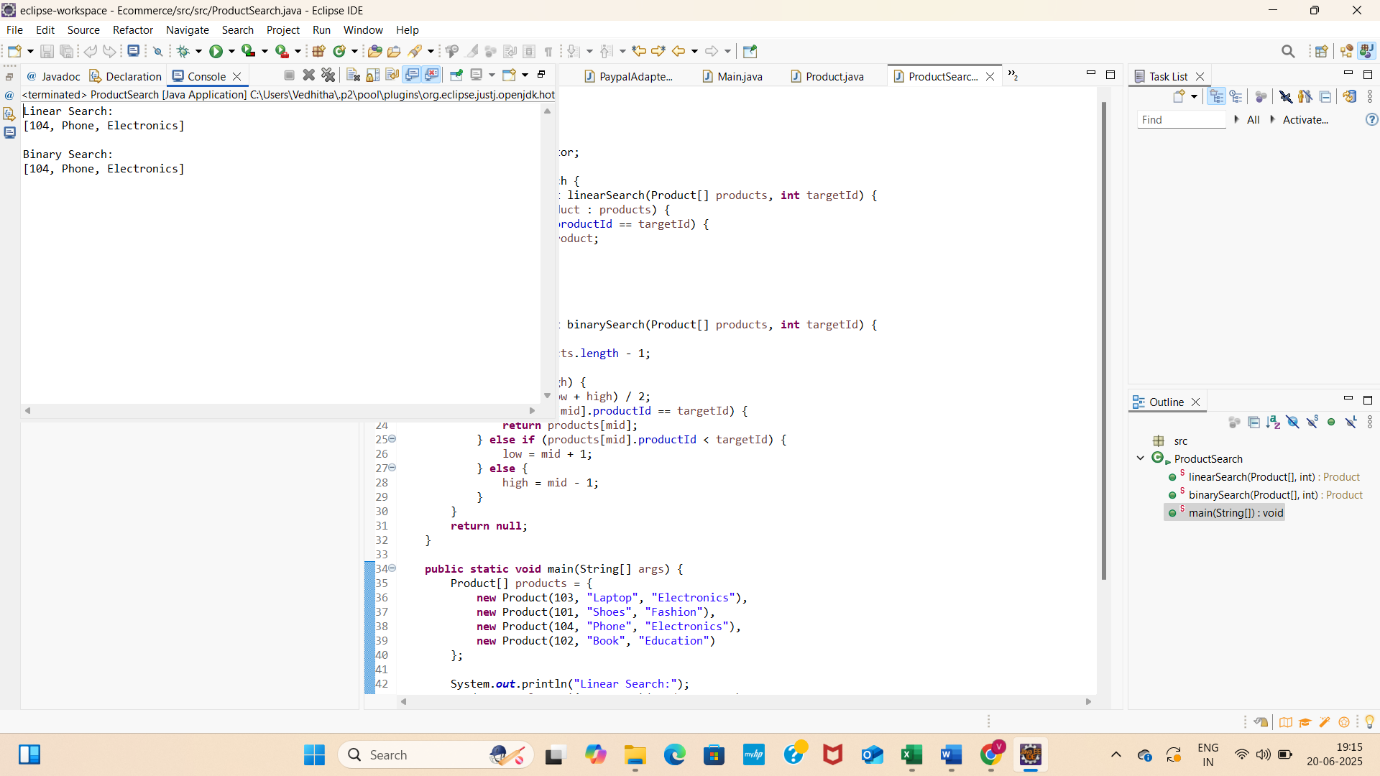
System.***out***.println("\nBinary Search:");

Product result2 = *binarySearch*(products, 104);

System.***out***.println(result2 != **null** ? result2 : "Product not found");

}

}



**Exercise 7: Financial Forecasting**

FinacialForecast.java

public class FinacialForecast {

    public static double futureValue(double presentValue, double rate, int years) {

        if (years == 0) {

            return presentValue;

        }

        return (1 + rate) \* futureValue(presentValue, rate, years - 1);

    }

    public static void main(String[] args) {

        double presentValue = 10000;

        double annualRate = 0.05;

        int years = 5;

        double result = futureValue(presentValue, annualRate, years);

        System.out.printf("Future Value after %d years: %.2f\n", years, result);

    }

}

