**WEEK1:**

**Exercise 1: Singleton Pattern**

public class SingletonPatternDemo {

static class Logger {

private static Logger instance;

private Logger() {

System.out.println("Logger instance created");

}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void log(String message) {

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

}

}

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("Starting the application");

Logger logger2 = Logger.getInstance();

logger2.log("Continuing the application");

if (logger1 == logger2) {

System.out.println("Both logger instances are the same (Singleton Verified)");

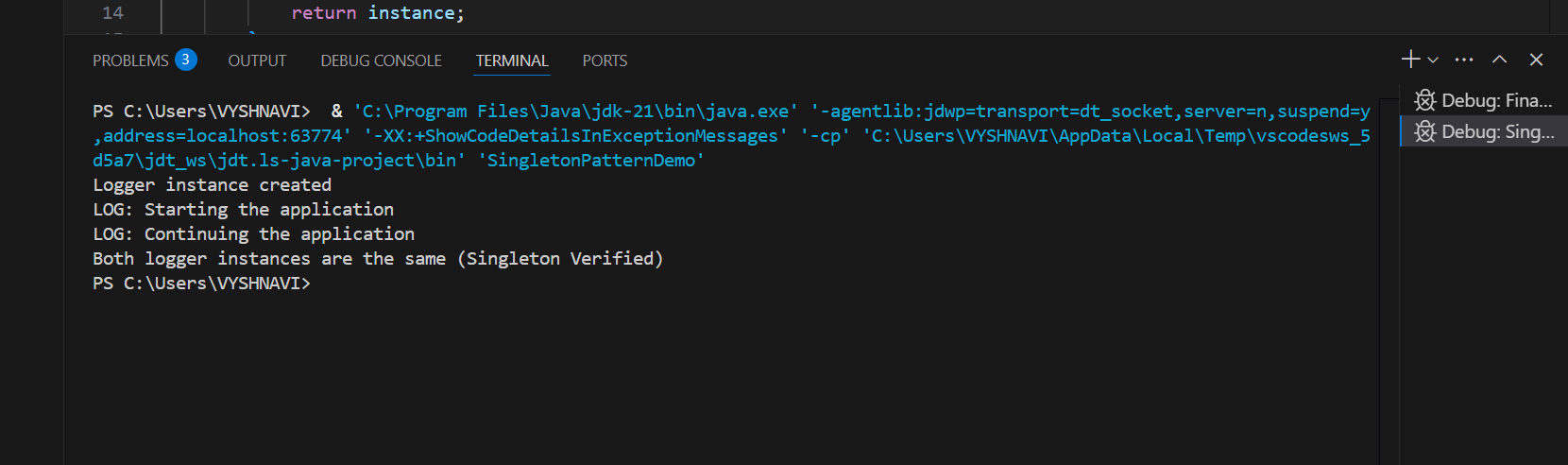
} else {

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

}

}

}



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

class ProductFactory {

    public static Product createProduct(String type) {

        switch (type.toLowerCase()) {

            case "mouse": return new Mouse();

            case "headphones": return new Headphones();

            default: throw new IllegalArgumentException("Unknown product type");

        }

    }

    public static void main(String[] args) {

        Product p1 = ProductFactory.createProduct("mouse");

        Product p2 = ProductFactory.createProduct("headphones");

        p1.use();

        p2.use();

    }

}

interface Product {

    void use();

}

class Mouse implements Product {

    public void use() {

        System.out.println("Using Wireless Mouse");

    }

}

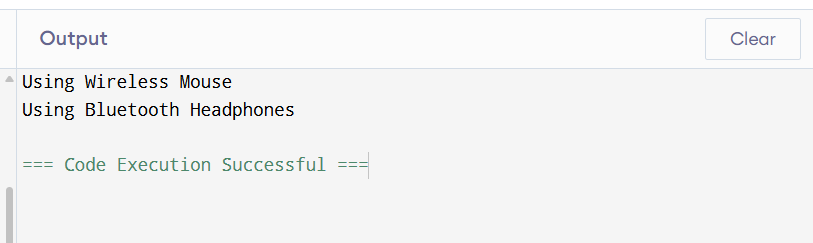
class Headphones implements Product {

    public void use() {

        System.out.println("Using Bluetooth Headphones");

    }

}



**Exercise 3: E-commerce Platform Search Function**

import java.util.\*;

public class ECommerceSearch {

    static class Product {

        String name, category;

        int price;

        Product(String name, String category, int price) {

            this.name = name;

            this.category = category;

            this.price = price;

        }

        public String toString() {

            return name + " (" + category + ") - ₹" + price;

        }

    }

    public static List<Product> search(List<Product> products, String keyword) {

        List<Product> result = new ArrayList<>();

        for (Product p : products) {

            if (keyword == null || p.name.toLowerCase().contains(keyword.toLowerCase()))

                result.add(p);

        }

        return result;

    }

    public static void main(String[] args) {

        List<Product> products = Arrays.asList(

            new Product("Wireless Mouse", "Electronics", 899),

            new Product("Running Shoes", "Footwear", 2599)

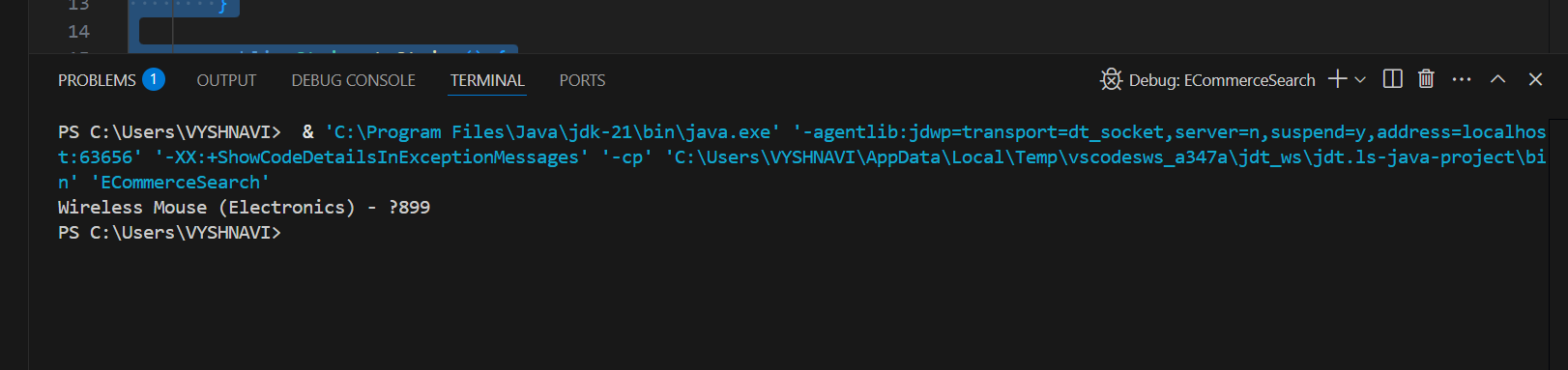
        );

        List<Product> result = search(products, "mouse");

        result.forEach(System.out::println);

    }

}



**Exercise 4: Financial Forecasting**

class Forecast {

static double predictRecursive(double initial, double rate, int years) {

if (years == 0)

return initial;

return predictRecursive(initial, rate, years - 1) \* (1 + rate);

}

static double predictIterative(double initial, double rate, int years) {

for (int i = 0; i < years; i++) {

initial \*= (1 + rate);

}

return initial;

}

}

public class FinancialForecasting {

public static void main(String[] args) {

double initialAmount = 10000;

double annualGrowthRate = 0.08;

int years = 5;

double futureValueRecursive = Forecast.predictRecursive(initialAmount, annualGrowthRate, years);

double futureValueIterative = Forecast.predictIterative(initialAmount, annualGrowthRate, years);

System.out.println("Future value using recursion: " + futureValueRecursive);

System.out.println("Future value using iteration: " + futureValueIterative);

}

}

