**Module 1(Design Patterns and Principles)**

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

**Solution Code:**

import java.util.Arrays;

import java.util.Comparator;

class Product {

    int productId;

    String productName;

    String category;

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

        this.productId = productId;

        this.productName = productName;

        this.category = category;

    }

}

public class ECommerceSearch {

    public static Product linearSearch(Product[] products, String targetName) {

        for (Product p : products) {

            if (p.productName.equalsIgnoreCase(targetName)) {

                return p;

            }

        }

        return null;

    }

    public static Product binarySearch(Product[] products, String targetName) {

        int low = 0;

        int high = products.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            int comparison = products[mid].productName.compareToIgnoreCase(targetName);

            if (comparison == 0) {

                return products[mid];

            } else if (comparison < 0) {

                low = mid + 1;

            } else {

                high = mid - 1;

            }

        }

        return null;

    }

    public static void main(String[] args) {

        Product[] products = {

            new Product(101, "Laptop", "Electronics"),

            new Product(102, "Shirt", "Clothing"),

            new Product(103, "Smartphone", "Electronics"),

            new Product(104, "Shoes", "Footwear"),

            new Product(105, "Book", "Stationery")

        };

        String searchTarget = "Shoes";

        Product linearResult = linearSearch(products, searchTarget);

        if (linearResult != null) {

            System.out.println("Found using Linear Search: " + linearResult.productName + " (ID: " + linearResult.productId + ")");

        } else {

            System.out.println("Product not found using Linear Search.");

        }

        Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));

        Product binaryResult = binarySearch(products, searchTarget);

        if (binaryResult != null) {

            System.out.println("Found using Binary Search: " + binaryResult.productName + " (ID: " + binaryResult.productId + ")");

        } else {

            System.out.println("Product not found using Binary Search.");

        }

    }

}

**Output (Exercise 2) :**

A black screen with white text

AI-generated content may be incorrect.

**Exercise 7: Financial Forecasting**

**Solution Code:**

public class FinancialForecast {

    public static double forecastRecursive(double initialValue, double growthRate, int years) {

        if (years == 0) return initialValue;

        return forecastRecursive(initialValue, growthRate, years - 1) \* (1 + growthRate);

    }

    public static double forecastOptimized(double initialValue, double growthRate, int years) {

        return initialValue \* Math.pow(1 + growthRate, years);

    }

    public static void main(String[] args) {

        double initial = 10000;

        double rate = 0.08;

        int years = 5;

        double valueRecursive = forecastRecursive(initial, rate, years);

        double valueOptimized = forecastOptimized(initial, rate, years);

        System.out.println("Predicted Value (Recursive): " + valueRecursive);

        System.out.println("Predicted Value (Optimized): " + valueOptimized);

    }

}

**Output (Exercise 7) :**

