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

**Scenario:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

**Product Class:**

package ds;

import java.util.\*;

public class Product implements Comparable<Product> {

int productId;

String productName;

String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

*@Override*

public int compareTo(Product other) {

return this.productName.compareToIgnoreCase(other.productName);

}

*@Override*

public String toString() {

return "[" + productId + "] " + productName + " (" + category + ")";

}

}

package ds;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.List;

public class ProductSearchList {

public static Product linearSearch(List<Product> products, String targetName) {

for (Product p : products) {

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

return p;

}

}

return null;

}

public static Product binarySearch(List<Product> products, String targetName) {

int low = 0, high = products.size() - 1;

while (low <= high) {

int mid = (low + high) / 2;

Product midProduct = products.get(mid);

int cmp = midProduct.productName.compareToIgnoreCase(targetName);

if (cmp == 0)

return midProduct;

else if (cmp < 0)

low = mid + 1;

else

high = mid - 1;

}

return null;

}

public static void main(String[] args) {

List<Product> products = new ArrayList<>(Arrays.*asList*(

new Product(1, "Laptop", "Electronics"),

new Product(2, "Shoes", "Fashion"),

new Product(3, "Watch", "Accessories"),

new Product(4, "Mobile", "Electronics"),

new Product(5, "Book", "Stationery")

));

System.***out***.println(" Unsorted Products ");

for (Product p : products) {

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

}

System.***out***.println("\nLinear Search");

Product found1 = *linearSearch*(products, "Mobile");

System.***out***.println(found1 != null ? found1 : "Product not found");

Collections.*sort*(products);

System.***out***.println("\nSorted Product");

for (Product p : products) {

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

}

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

Product found2 = *binarySearch*(products, "Mobile");

System.***out***.println(found2 != null ? found2 : "Product not found");

}

}

**Output:**A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 7: Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

package ds;

import java.util.Scanner;

public class Forcast {

//normal recursion method

public static double calculateFutureValue(double principal, double rate, int years) {

if (years == 0) {

return principal;

}

return *calculateFutureValue*(principal, rate, years - 1) \* (1 + rate);

}

// optimized method

public static double calculateFutureValueOpt(double principal, double rate, int years, double[] memo) {

if (years == 0) {

return principal;

}

if (memo[years] != 0) {

return memo[years];

}

memo[years] = *calculateFutureValueOpt*(principal, rate, years - 1, memo) \* (1 + rate);

return memo[years];

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

System.***out***.print("Enter principal amount: ");

double principal = scanner.nextDouble();

System.***out***.print("Enter growth rate : ");

double rate = scanner.nextDouble();

System.***out***.print("Enter number of years: ");

int years = scanner.nextInt();

// Normal recursion

double futureValue = *calculateFutureValue*(principal, rate, years);

System.***out***.printf("Future value after %d years (Normal Recursion): %.2f%n", years, futureValue);

//optimized recursion

double[] storeprev = new double[years + 1];

double futureValueopt = *calculateFutureValueOpt*(principal, rate, years,storeprev);

System.***out***.printf("Future value after %d years (Optimized Recursion): %.2f%n", years, futureValueopt);

scanner.close();

}

}

OUTPUT:

