**Exercise 1: E-commerce Platform Search Function**

**SearchDemo.java**

package EcommercePlatformSearcFunction;

import java.util.Arrays;

import java.util.Comparator;

public class SearchDemo {

public static void main(String[] args) {

Product[] products = {

new Product(3, "Laptop", "Electronics"),

new Product(1, "Book", "Stationery"),

new Product(2, "Smartphone", "Electronics")

};

// Linear Search

Product resultLinear = *linearSearch*(products, 2);

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

// Sort products by productId for Binary Search

Arrays.*sort*(products, Comparator.*comparingInt*(Product::getProductId));

// Binary Search

Product resultBinary = *binarySearch*(products, 2);

System.*out*.println("Binary Search Result: " + resultBinary);

}

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

for (Product product : products) {

if (product.getProductId() == targetId) {

return product;

}

}

return null;

}

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

int left = 0, right = products.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

int midId = products[mid].getProductId();

if (midId == targetId) {

return products[mid];

} else if (midId < targetId) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

}

**Product.java**

package EcommercePlatformSearcFunction;

public class Product {

private int productId;

private String productName;

private String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

public int getProductId() {

return productId;

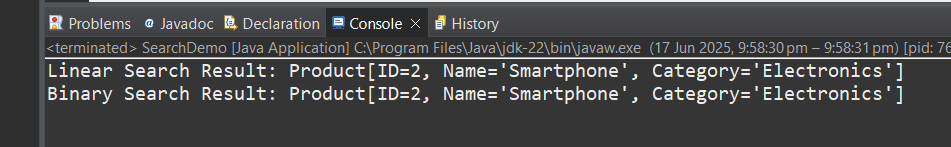
}

public String toString() {

return "Product[ID=" + productId + ", Name='" + productName + "', Category='" + category + "']";

}

}

****

**Exercise 2: Financial Forecasting**

package FinancialForecasting;

public class FinancialForecast {

// Recursive approach

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

if (years == 0) {

return principal; // Base Case

}

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

}

// Iterative approach

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

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

principal \*= (1 + rate);

}

return principal;

}

// Direct formula approach (Best)

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

return principal \* Math.*pow*((1 + rate), years);

}

public static void main(String[] args) {

double initialInvestment = 1000.0; // Starting value

double annualGrowthRate = 0.05; // 5% growth rate

int numberOfYears = 5; // Predict for 5 years

double valueRecursive = *futureValueRecursive*(initialInvestment, annualGrowthRate, numberOfYears);

double valueIterative = *futureValueIterative*(initialInvestment, annualGrowthRate, numberOfYears);

double valueFormula = *futureValueFormula*(initialInvestment, annualGrowthRate, numberOfYears);

System.***out***.printf("Future Value (Recursive): %.2f\n", valueRecursive);

System.***out***.printf("Future Value (Iterative): %.2f\n", valueIterative);

System.***out***.printf("Future Value (Formula) : %.2f\n", valueFormula);

}

}

