**WEEK-1**

**DATA STRUCTURES AND ALGORITHMS**

**Exercise 1: Inventory Management System**

import java.util.HashMap;

class Product {

public int productId;

public String productName;

public int quantity;

public int price;

static HashMap<Integer, Product> inventory = new HashMap<>();

public Product(int productId, String productName, int quantity, int price) {

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

public static void add(Product product) {

inventory.put(product.productId, product);

System.out.println("Added: " + product.productName);

}

public static void update(int productId, int newQuantity, int newPrice) {

Product p = inventory.get(productId);

if (p != null) {

p.quantity = newQuantity;

p.price = newPrice;

System.out.println("Updated: " + p.productName);

} else {

System.out.println("Product ID not found!");

}

}

public static void delete(int productId) {

Product removed = inventory.remove(productId);

if (removed != null) {

System.out.println("Deleted: " + removed.productName);

} else {

System.out.println("Product ID not found!");

}

}

public static void viewAll() {

if (inventory.isEmpty()) {

System.out.println("Inventory is empty.");

} else {

for (Product p : inventory.values()) {

System.out.println("ID: " + p.productId + ", Name: " + p.productName + ", Qty: " + p.quantity + ", Price: ₹" + p.price);

}

}

}

}

public class Main {

public static void main(String[] args) {

Product p1 = new Product(101, "Laptop", 10, 75000);

Product p2 = new Product(102, "Mouse", 50, 700);

Product p3 = new Product(103, "Keyboard", 50, 1200);

Product.add(p1);

Product.add(p2);

Product.add(p3);

Product.update(102, 60, 650);

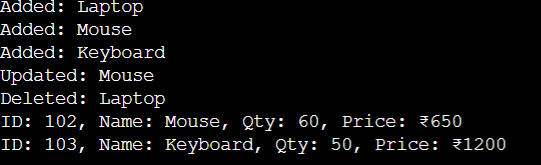
Product.delete(101);

Product.viewAll();

}

}

**Output:**



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

import java.util.\*;

class Product {

public int productId;

public String productName;

public String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String toString() {

return "ID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

public class Main {

public static void main(String[] args) {

// Array for linear search (unsorted)

Product[] linear\_arr = {

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

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

new Product(3, "Book", "Stationery")

};

Product[] binary\_arr = linear\_arr.clone();

Arrays.sort(binary\_arr, (a, b) -> a.productName.compareToIgnoreCase(b.productName));

Product foundLinear = linearSearch(linear\_arr, "Shoes");

System.out.println(foundLinear != null ? "Found (Linear): " + foundLinear : "Not Found (Linear)");

Product foundBinary = binarySearch(binary\_arr, "Shoes");

System.out.println(foundBinary != null ? "Found (Binary): " + foundBinary : "Not Found (Binary)");

}

public static Product linearSearch(Product[] arr, String name) {

for (Product p : arr) {

if (p.productName.equalsIgnoreCase(name)) return p;

}

return null;}

public static Product binarySearch(Product[] arr, String name) {

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

while (left <= right) {

int mid = (left + right) / 2;

int cmp = arr[mid].productName.compareToIgnoreCase(name);

if (cmp == 0) return arr[mid];

else if (cmp < 0) left = mid + 1;

else right = mid - 1;

}

return null;

}

}

**OUTPUT:**



**Exercise 7: Financial Forecasting:**

import java.util.List;

import java.util.ArrayList;

import java.util.Map;

import java.util.HashMap;

public class FinancialForecasting {

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

if (years == 0) return principal;

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

}

public double calculateFutureValueOptimized(double principal, double rate, int years, Map<String, Double> cache) {

String key = principal + ":" + rate + ":" + years;

if (cache.containsKey(key)) return cache.get(key);

if (years == 0) return principal;

double result = calculateFutureValueOptimized(principal \* (1 + rate), rate, years - 1, cache);

cache.put(key, result);

return result;

}

public List<Double> generateForecast(double initialValue, double growthRate, int periods) {

List<Double> forecast = new ArrayList<>();

generateForecastRecursive(initialValue, growthRate, periods, forecast);

return forecast;

}

private void generateForecastRecursive(double value, double rate, int remaining, List<Double> forecast) {

forecast.add(value);

if (remaining > 0) {

generateForecastRecursive(value \* (1 + rate), rate, remaining - 1, forecast);

}

}

public static void main(String[] args) {

FinancialForecasting tool = new FinancialForecasting();

double initial = 1000.0;

double rate = 0.05;

int years = 5;

double result = tool.calculateFutureValue(initial, rate, years);

System.out.println("Future Value (Recursive): " + result);

double optimizedResult = tool.calculateFutureValueOptimized(initial, rate, years, new HashMap<>());

System.out.println("Future Value (Optimized): " + optimizedResult);

List<Double> forecast = tool.generateForecast(initial, rate, years);

System.out.println("Forecast:");

for (int i = 0; i < forecast.size(); i++) {

System.out.println("Year " + i + ": " + forecast.get(i));

}

}

}

**OUTPUT:**

