**Exercise 1: Inventory Management System**

import java.util.HashMap;

class Product {

int productId;

String productName;

int quantity;

double price;

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

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

public String toString() {

return productId + " - " + productName + " - " + quantity + " - ₹" + price;

}

}

class Inventory {

HashMap<Integer, Product> products = new HashMap<>();

public void addProduct(Product p) {

products.put(p.productId, p);

}

public void updateProduct(int id, int quantity, double price) {

if (products.containsKey(id)) {

Product p = products.get(id);

p.quantity = quantity;

p.price = price;

}

}

public void deleteProduct(int id) {

products.remove(id);

}

public void printInventory() {

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

System.out.println(p);

}

}

}

public class Main1 {

public static void main(String[] args) {

Inventory inv = new Inventory();

inv.addProduct(new Product(101, "Laptop", 10, 55000));

inv.addProduct(new Product(102, "Mouse", 50, 500));

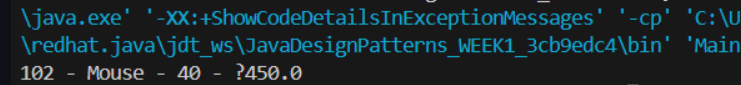
inv.updateProduct(102, 40, 450);

inv.deleteProduct(101);

inv.printInventory();

}

}



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

class Product1 {

int productId;

String productName;

int quantity;

double price;

public Product1(int productId, String productName, int quantity, double price) {

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

public String toString() {

return productId + " - " + productName + " - Qty: " + quantity + " - ₹" + price;

}

}

public class Main2 {

public static Product1 linearSearch(Product1[] products, String name) {

for (Product1 p : products) {

if (p.productName.equalsIgnoreCase(name))

return p;

}

return null;

}

public static Product1 binarySearch(Product1[] products, String name) {

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

while (left <= right) {

int mid = (left + right) / 2;

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

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

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

else right = mid - 1;

}

return null;

}

public static void main(String[] args) {

Product1[] products = {

new Product1(101, "Keyboard", 20, 1500.0),

new Product1(102, "Laptop", 10, 55000.0),

new Product1(103, "Mouse", 100, 500.0),

};

// Binary search requires sorting by name

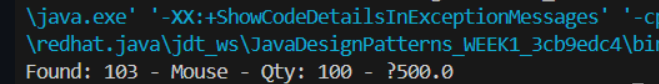
java.util.Arrays.sort(products, (a, b) -> a.productName.compareToIgnoreCase(b.productName));

Product1 found = binarySearch(products, "Mouse");

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

}

}

****

**Exercise 3: Sorting Customer Orders**

class Order {

int orderId;

String customerName;

double totalPrice;

public Order(int orderId, String customerName, double totalPrice) {

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public String toString() {

return orderId + " - " + customerName + " - ₹" + totalPrice;

}

}

public class Main3 {

public static void bubbleSort(Order[] orders) {

int n = orders.length;

for (int i = 0; i < n-1; i++) {

for (int j = 0; j < n-i-1; j++) {

if (orders[j].totalPrice > orders[j+1].totalPrice) {

Order temp = orders[j];

orders[j] = orders[j+1];

orders[j+1] = temp;

}

}

}

}

public static void quickSort(Order[] orders, int low, int high) {

if (low < high) {

int pi = partition(orders, low, high);

quickSort(orders, low, pi - 1);

quickSort(orders, pi + 1, high);

}

}

private static int partition(Order[] orders, int low, int high) {

double pivot = orders[high].totalPrice;

int i = low - 1;

for (int j = low; j < high; j++) {

if (orders[j].totalPrice < pivot) {

i++;

Order temp = orders[i];

orders[i] = orders[j];

orders[j] = temp;

}

}

Order temp = orders[i + 1];

orders[i + 1] = orders[high];

orders[high] = temp;

return i + 1;

}

public static void main(String[] args) {

Order[] orders = {

new Order(201, "Alice", 1500),

new Order(202, "Bob", 700),

new Order(203, "Charlie", 2500),

};

quickSort(orders, 0, orders.length - 1);

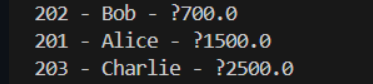
for (Order o : orders) {

System.out.println(o);

}

}

}

****

**Exercise 4: Employee Management System**

class Employee {

int employeeId;

String name;

String position;

double salary;

public Employee(int id, String name, String position, double salary) {

this.employeeId = id;

this.name = name;

this.position = position;

this.salary = salary;

}

public String toString() {

return employeeId + " - " + name + " - " + position + " - ₹" + salary;

}

}

public class Main4 {

static Employee[] employees = new Employee[100];

static int count = 0;

public static void addEmployee(Employee emp) {

employees[count++] = emp;

}

public static Employee searchEmployee(int id) {

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

if (employees[i].employeeId == id)

return employees[i];

}

return null;

}

public static void deleteEmployee(int id) {

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

if (employees[i].employeeId == id) {

for (int j = i; j < count - 1; j++) {

employees[j] = employees[j + 1];

}

employees[--count] = null;

return;

}

}

}

public static void printEmployees() {

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

System.out.println(employees[i]);

}

}

public static void main(String[] args) {

addEmployee(new Employee(1, "John", "Manager", 50000));

addEmployee(new Employee(2, "Jane", "HR", 40000));

deleteEmployee(1);

printEmployees();

}

}

****

**Exercise 5: Task Management System**

class Task {

int taskId;

String taskName;

String status;

Task next;

public Task(int id, String name, String status) {

this.taskId = id;

this.taskName = name;

this.status = status;

this.next = null;

}

public String toString() {

return taskId + " - " + taskName + " [" + status + "]";

}

}

class TaskManager {

Task head;

public void addTask(Task newTask) {

if (head == null) {

head = newTask;

} else {

Task temp = head;

while (temp.next != null)

temp = temp.next;

temp.next = newTask;

}

}

public void deleteTask(int id) {

if (head == null) return;

if (head.taskId == id) {

head = head.next;

return;

}

Task prev = null, curr = head;

while (curr != null && curr.taskId != id) {

prev = curr;

curr = curr.next;

}

if (curr != null) prev.next = curr.next;

}

public Task searchTask(int id) {

Task temp = head;

while (temp != null) {

if (temp.taskId == id) return temp;

temp = temp.next;

}

return null;

}

public void printTasks() {

Task temp = head;

while (temp != null) {

System.out.println(temp);

temp = temp.next;

}

}

}

public class Main5 {

public static void main(String[] args) {

TaskManager tm = new TaskManager();

tm.addTask(new Task(1, "Design UI", "Pending"));

tm.addTask(new Task(2, "Implement Backend", "In Progress"));

tm.deleteTask(1);

tm.printTasks();

}

}



**Exercise 6: Library Management System**

import java.util.Arrays;

class Book {

int bookId;

String title;

String author;

public Book(int id, String title, String author) {

this.bookId = id;

this.title = title;

this.author = author;

}

public String toString() {

return bookId + " - " + title + " by " + author;

}

}

public class Main6 {

public static Book linearSearch(Book[] books, String title) {

for (Book b : books) {

if (b.title.equalsIgnoreCase(title))

return b;

}

return null;

}

public static Book binarySearch(Book[] books, String title) {

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

while (left <= right) {

int mid = (left + right) / 2;

int cmp = books[mid].title.compareToIgnoreCase(title);

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

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

else right = mid - 1;

}

return null;

}

public static void main(String[] args) {

Book[] books = {

new Book(101, "Algorithms", "Cormen"),

new Book(102, "Java Programming", "James Gosling"),

new Book(103, "Data Structures", "Weiss")

};

Arrays.sort(books, (a, b) -> a.title.compareToIgnoreCase(b.title));

Book found = binarySearch(books, "Java Programming");

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

}

}

****

**Exercise 7: Financial Forecasting**

public class Main7 {

// Recursive method: futureValue = currentValue \* (1 + growthRate)^years

public static double predictFutureValue(double currentValue, double growthRate, int years) {

if (years == 0) return currentValue;

return (1 + growthRate) \* predictFutureValue(currentValue, growthRate, years - 1);

}

public static void main(String[] args) {

double currentValue = 10000;

double growthRate = 0.10; // 10% growth

int years = 5;

double futureValue = predictFutureValue(currentValue, growthRate, years);

System.out.printf("Future Value after %d years = ₹%.2f\n", years, futureValue);

}

}

