**Exercise 1: Inventory Management System**

Code:-

import java.util.HashMap;

public class Product {

private int productId;

private String productName;

private int quantity;

private double price;

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

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

// Getters and setters

public int getProductId() { return productId; }

public String getProductName() { return productName; }

public int getQuantity() { return quantity; }

public double getPrice() { return price; }

public void setProductName(String productName) { this.productName = productName; }

public void setQuantity(int quantity) { this.quantity = quantity; }

public void setPrice(double price) { this.price = price; }

@Override

public String toString() {

return "Product{" +

"ID=" + productId +

", Name='" + productName + '\'' +

", Quantity=" + quantity +

", Price=" + price +

'}';

}

}

public class InventoryManager {

private HashMap<Integer, Product> inventory;

public InventoryManager() {

inventory = new HashMap<>();

}

// Add Product

public void addProduct(Product product) {

inventory.put(product.getProductId(), product);

System.out.println("Product added: " + product);

}

// Update Product

public void updateProduct(int productId, String name, int quantity, double price) {

Product product = inventory.get(productId);

if (product != null) {

product.setProductName(name);

product.setQuantity(quantity);

product.setPrice(price);

System.out.println("Product updated: " + product);

} else {

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

}

}

// Delete Product

public void deleteProduct(int productId) {

Product removed = inventory.remove(productId);

if (removed != null) {

System.out.println("Product removed: " + removed);

} else {

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

}

}

// Display Inventory

public void displayInventory() {

if (inventory.isEmpty()) {

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

} else {

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

System.out.println(product);

}

}

}

}

public class Main {

public static void main(String[] args) {

InventoryManager manager = new InventoryManager();

// Adding Products

manager.addProduct(new Product(101, "Laptop", 10, 55000.0));

manager.addProduct(new Product(102, "Keyboard", 50, 800.0));

// Updating a Product

manager.updateProduct(101, "Gaming Laptop", 8, 75000.0);

// Deleting a Product

manager.deleteProduct(102);

// Displaying Inventory

manager.displayInventory();

}

}

Output:-

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

Code:-

import java.util.\*;

class 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 String toString() {

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

}

}

public class ProductSearch {

// Linear Search

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

for (Product p : products) {

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

return p;

}

}

return null;

}

// Binary Search (Array must be sorted by productName)

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

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

while (left <= right) {

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

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

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) {

Product[] products = {

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

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

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

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

new Product(5, "Bag", "Fashion")

};

// Linear Search

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

Product result1 = linearSearch(products, "Phone");

System.out.println(result1 != null ? result1 : "Product not found");

// Sort for Binary Search

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

// Binary Search

System.out.println("\nBinary Search Result:");

Product result2 = binarySearch(products, "Phone");

System.out.println(result2 != null ? result2 : "Product not found");

}

}

Output:-

**Exercise 3: Sorting Customer Orders**

Code:-

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;

}

@Override

public String toString() {

return "Order{" +

"orderId=" + orderId +

", customerName='" + customerName + '\'' +

", totalPrice=" + totalPrice +

'}';

}

}

public class OrderSorter {

public static void main(String[] args) {

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) {

// Swap

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);

}

}

public 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++;

// Swap

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;

}

Order[] orders = {

new Order(101, "Alice", 299.99),

new Order(102, "Bob", 159.49),

new Order(103, "Charlie", 499.00),

new Order(104, "Diana", 99.99)

};

Order[] bubbleSortedOrders = orders.clone();

bubbleSort(bubbleSortedOrders);

System.out.println("Sorted by Bubble Sort:");

for (Order o : bubbleSortedOrders) {

System.out.println(o);

}

Order[] quickSortedOrders = orders.clone();

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

System.out.println("\nSorted by Quick Sort:");

for (Order o : quickSortedOrders) {

System.out.println(o);

}

}

}

Output:-

**Exercise 4: Employee Management System**

Code:-

class Employee {

int employeeId;

String name;

String position;

double salary;

Employee(int employeeId, String name, String position, double salary) {

this.employeeId = employeeId;

this.name = name;

this.position = position;

this.salary = salary;

}

public void display() {

System.out.println("ID: " + employeeId + ", Name: " + name +

", Position: " + position + ", Salary: " + salary);

}

}

public class EmployeeManagementSystem {

private Employee[] employees;

private int size;

public EmployeeManagementSystem(int capacity) {

employees = new Employee[capacity];

size = 0;

}

// Add employee

public void addEmployee(Employee emp) {

if (size < employees.length) {

employees[size++] = emp;

} else {

System.out.println("Cannot add more employees. Array is full.");

}

}

// Search employee by ID

public Employee searchEmployee(int id) {

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

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

return employees[i];

}

}

return null;

}

// Traverse all employees

public void displayAllEmployees() {

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

employees[i].display();

}

}

// Delete employee by ID

public void deleteEmployee(int id) {

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

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

// Shift remaining elements left

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

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

}

employees[size - 1] = null;

size--;

System.out.println("Employee with ID " + id + " deleted.");

return;

}

}

System.out.println("Employee with ID " + id + " not found.");

}

// Main method for testing

public static void main(String[] args) {

EmployeeManagementSystem system = new EmployeeManagementSystem(5);

system.addEmployee(new Employee(101, "Alice", "Manager", 75000));

system.addEmployee(new Employee(102, "Bob", "Engineer", 60000));

system.addEmployee(new Employee(103, "Charlie", "Clerk", 35000));

System.out.println("All Employees:");

system.displayAllEmployees();

System.out.println("\nSearching for employee ID 102:");

Employee found = system.searchEmployee(102);

if (found != null) found.display();

System.out.println("\nDeleting employee ID 102:");

system.deleteEmployee(102);

System.out.println("\nAll Employees after deletion:");

system.displayAllEmployees();

}

}

Output:-

**Exercise 5: Task Management System**

Code:-

class Task {

int taskId;

String taskName;

String status; // e.g., "Pending", "Completed"

Task(int taskId, String taskName, String status) {

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

}

}

class Node {

Task task;

Node next;

Node(Task task) {

this.task = task;

this.next = null;

}

}

class TaskManager {

private Node head;

// Add task at the end

public void addTask(Task task) {

Node newNode = new Node(task);

if (head == null) {

head = newNode;

return;

}

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

// Search task by taskId

public Task searchTask(int taskId) {

Node current = head;

while (current != null) {

if (current.task.taskId == taskId) {

return current.task;

}

current = current.next;

}

return null; // Not found

}

// Delete task by taskId

public boolean deleteTask(int taskId) {

if (head == null) return false;

if (head.task.taskId == taskId) {

head = head.next;

return true;

}

Node current = head;

while (current.next != null && current.next.task.taskId != taskId) {

current = current.next;

}

if (current.next != null) {

current.next = current.next.next;

return true;

}

return false; // Not found

}

// Traverse and print all tasks

public void displayTasks() {

Node current = head;

while (current != null) {

Task t = current.task;

System.out.println("ID: " + t.taskId + ", Name: " + t.taskName + ", Status: " + t.status);

current = current.next;

}

}

}

public class Main {

public static void main(String[] args) {

TaskManager manager = new TaskManager();

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

manager.addTask(new Task(2, "Build Backend", "In Progress"));

manager.addTask(new Task(3, "Testing", "Pending"));

System.out.println("All Tasks:");

manager.displayTasks();

System.out.println("\nSearching for Task ID 2:");

Task found = manager.searchTask(2);

if (found != null) {

System.out.println("Found: " + found.taskName);

}

System.out.println("\nDeleting Task ID 1...");

manager.deleteTask(1);

System.out.println("\nAll Tasks After Deletion:");

manager.displayTasks();

}

}

Output:-

**Exercise 6: Library Management System**

Code:-

import java.util.\*;

class Book {

int bookId;

String title;

String author;

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

this.bookId = bookId;

this.title = title;

this.author = author;

}

@Override

public String toString() {

return "BookId: " + bookId + ", Title: " + title + ", Author: " + author;

}

}

public class LibraryManager {

// Linear Search by Title

public static List<Book> linearSearchByTitle(List<Book> books, String searchTitle) {

List<Book> results = new ArrayList<>();

for (Book book : books) {

if (book.title.equalsIgnoreCase(searchTitle)) {

results.add(book);

}

}

return results;

}

// Binary Search by Title (Assumes books sorted by title)

public static Book binarySearchByTitle(List<Book> books, String searchTitle) {

int left = 0, right = books.size() - 1;

while (left <= right) {

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

Book midBook = books.get(mid);

int compare = midBook.title.compareToIgnoreCase(searchTitle);

if (compare == 0)

return midBook;

else if (compare < 0)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

public static void main(String[] args) {

List<Book> books = new ArrayList<>(List.of(

new Book(1, "The Alchemist", "Paulo Coelho"),

new Book(2, "Atomic Habits", "James Clear"),

new Book(3, "1984", "George Orwell"),

new Book(4, "To Kill a Mockingbird", "Harper Lee"),

new Book(5, "The Great Gatsby", "F. Scott Fitzgerald")

));

// Linear Search

System.out.println("Linear Search:");

List<Book> foundBooks = linearSearchByTitle(books, "1984");

foundBooks.forEach(System.out::println);

// Binary Search requires sorted list

books.sort(Comparator.comparing(b -> b.title.toLowerCase()));

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

Book foundBook = binarySearchByTitle(books, "1984");

System.out.println(foundBook != null ? foundBook : "Book not found.");

}

}

Output:-