Assignment 1: Data Structures and Algorithms.

**Exercise 1: Inventory Management System**

**Code:**

import java.util.HashMap;

class Product {

private String productId;

private String productName;

private int quantity;

private double price;

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

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

// Getters and Setters

public String getProductId() {

return productId;

}

public String getProductName() {

return productName;

}

public int getQuantity() {

return quantity;

}

public double getPrice() {

return price;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public void setPrice(double price) {

this.price = price;

}

public String toString() {

return "Product{" +

"productId='" + productId + '\'' +

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

", quantity=" + quantity +

", price=" + price +

'}';

}

}

class InventoryManager {

private HashMap<String, Product> inventory;

public InventoryManager() {

this.inventory = new HashMap<>();

}

public void addProduct(Product product) {

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

}

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

Product product = inventory.get(productId);

if (product != null) {

product.setQuantity(quantity);

product.setPrice(price);

} else {

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

}

}

public void deleteProduct(String productId) {

inventory.remove(productId);

}

public void displayProducts() {

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

System.out.println(product);

}

}

}

class Main {

public static void main(String[] args) {

InventoryManager inventoryManager = new InventoryManager();

Product product1 = new Product("P001", "Air pods", 10, 99.00);

Product product2 = new Product("P002", "Smartphone", 20, 499.99);

inventoryManager.addProduct(product1);

inventoryManager.addProduct(product2);

System.out.println("Current Inventory:");

inventoryManager.displayProducts();

inventoryManager.updateProduct("P001", 8, 899.99);

System.out.println("\nInventory after update:");

inventoryManager.displayProducts();

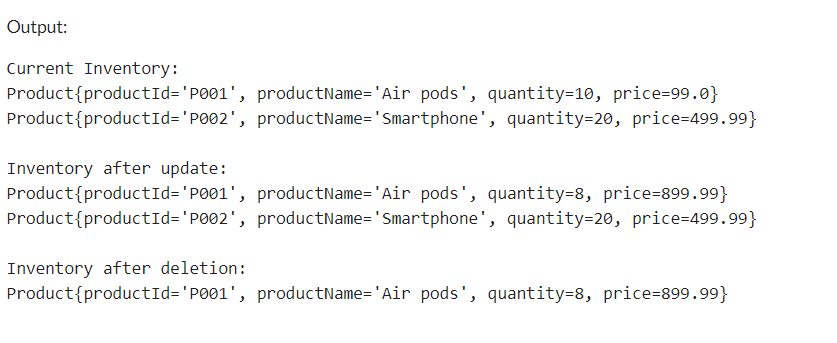
inventoryManager.deleteProduct("P002");

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

inventoryManager.displayProducts();

}

}



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

**Code:**

import java.util.Arrays;

class Product {

private String productId;

private String productName;

private String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String getProductId() {

return productId;

}

public String getProductName() {

return productName;

}

public String getCategory() {

return category;

}

public String toString() {

return "Product{" +

"productId='" + productId + '\'' +

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

", category='" + category + '\'' +

'}';

}

}

class SearchAlgorithms {

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

for (Product product : products) {

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

return product;

}

}

return null;

}

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

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

if (products[mid].getProductId().equals(targetId)) {

return products[mid]; // Product found

}

if (products[mid].getProductId().compareTo(targetId) < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null; // Product not found

}

}

class Main {

public static void main(String[] args) {

Product[] products = {

new Product("P001", "Laptop", "Electronics"),

new Product("P002", "Smartphone", "Electronics"),

new Product("P003", "Tablet", "Electronics"),

new Product("P004", "Headphones", "Accessories"),

new Product("P005", "Smartwatch", "Wearables")

};

Arrays.sort(products, (a, b) -> a.getProductId().compareTo(b.getProductId()));

String targetId = "P003";

Product foundProductLinear = SearchAlgorithms.linearSearch(products, targetId);

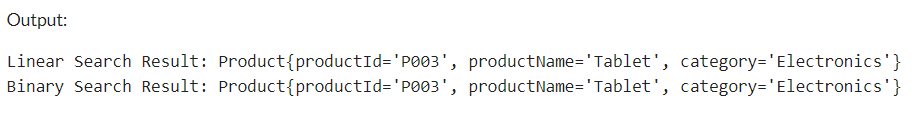
System.out.println("Linear Search Result: " + (foundProductLinear != null ? foundProductLinear : "Not Found"));

Product foundProductBinary = SearchAlgorithms.binarySearch(products, targetId);

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

}

}



**Exercise 3: Sorting Customer Orders**

**Code:**

class Order {

private String orderId;

private String customerName;

private double totalPrice;

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

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public String getOrderId() {

return orderId;

}

public String getCustomerName() {

return customerName;

}

public double getTotalPrice() {

return totalPrice;

}

}

class SortService {

public static void bubbleSort(Order[] orders) {

int n = orders.length;

boolean swapped;

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

swapped = false;

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

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

// Swap orders[j] and orders[j + 1]

Order temp = orders[j];

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

orders[j + 1] = temp;

swapped = true;

}

}

if (!swapped) break;

}

}

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

if (low < high) {

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

quickSort(orders, low, pi - 1); // Sort elements before partition

quickSort(orders, pi + 1, high); // Sort elements after partition

}

}

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

double pivot = orders[high].getTotalPrice(); // Pivot

int i = (low - 1); // Index of smaller element

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

// If current element is smaller than or equal to pivot

if (orders[j].getTotalPrice() <= pivot) {

i++;

// Swap orders[i] and orders[j]

Order temp = orders[i];

orders[i] = orders[j];

orders[j] = temp;

}

}

// Swap orders[i + 1] and orders[high] (or pivot)

Order temp = orders[i + 1];

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

orders[high] = temp;

return i + 1;

}

}

class Main {

public static void main(String[] args) {

Order[] orders = {

new Order("001", "Alice", 250.00),

new Order("002", "Bob", 150.00),

new Order("003", "Charlie", 300.00)

};

// Bubble Sort

System.out.println("Sorting orders using Bubble Sort:");

SortService.bubbleSort(orders);

for (Order order : orders) {

System.out.println(order.getOrderId() + ": " + order.getTotalPrice());

}

// Resetting orders for Quick Sort

orders = new Order[]{

new Order("001", "Alice", 250.00),

new Order("002", "Bob", 150.00),

new Order("003", "Charlie", 300.00)

};

// Quick Sort

System.out.println("\nSorting orders using Quick Sort:");

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

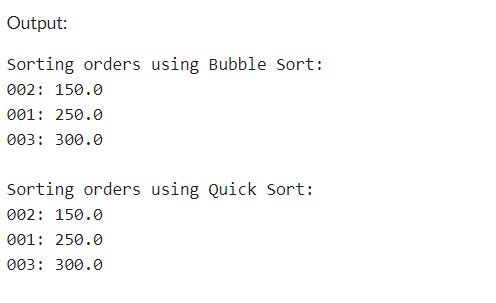
for (Order order : orders) {

System.out.println(order.getOrderId() + ": " + order.getTotalPrice());

}

}

}



**Exercise 4: Employee Management System**

**Code:**

class Employee {

private String employeeId;

private String name;

private String position;

private double salary;

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

this.employeeId = employeeId;

this.name = name;

this.position = position;

this.salary = salary;

}

public String getEmployeeId() {

return employeeId;

}

public String getName() {

return name;

}

public String getPosition() {

return position;

}

public double getSalary() {

return salary;

}

public String toString() {

return "Employee{" +

"employeeId='" + employeeId + '\'' +

", name='" + name + '\'' +

", position='" + position + '\'' +

", salary=" + salary +

'}';

}

}

class EmployeeManagement {

private Employee[] employees;

private int size; // Current number of employees

public EmployeeManagement(int capacity) {

employees = new Employee[capacity];

size = 0;

}

// Method to add an employee

public void addEmployee(Employee employee) {

if (size < employees.length) {

employees[size] = employee;

size++;

} else {

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

}

}

// Method to search for an employee by employeeId

public Employee searchEmployee(String employeeId) {

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

if (employees[i].getEmployeeId().equals(employeeId)) {

return employees[i]; // Employee found

}

}

return null; // Employee not found

}

// Method to traverse and display all employees

public void traverseEmployees() {

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

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

}

}

// Method to delete an employee by employeeId

public boolean deleteEmployee(String employeeId) {

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

if (employees[i].getEmployeeId().equals(employeeId)) {

// Shift elements to the left to fill the gap

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

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

}

employees[size - 1] = null; // Clear the last element

size--;

return true; // Employee deleted

}

}

return false; // Employee not found

}

}

class Main {

public static void main(String[] args) {

EmployeeManagement management = new EmployeeManagement(5);

// Adding employees

management.addEmployee(new Employee("E001", "Alice", "Developer", 70000));

management.addEmployee(new Employee("E002", "Bob", "Manager", 80000));

management.addEmployee(new Employee("E003", "Charlie", "Designer", 60000));

// Traversing employees

System.out.println("Employee List:");

management.traverseEmployees();

// Searching for an employee

Employee foundEmployee = management.searchEmployee("E002");

System.out.println("\nSearching for E002:");

System.out.println(foundEmployee != null ? foundEmployee : "Employee not found");

// Deleting an employee

boolean deleted = management.deleteEmployee("E001");

System.out.println("\nDeleting E001: " + (deleted ? "Success" : "Employee not found"));

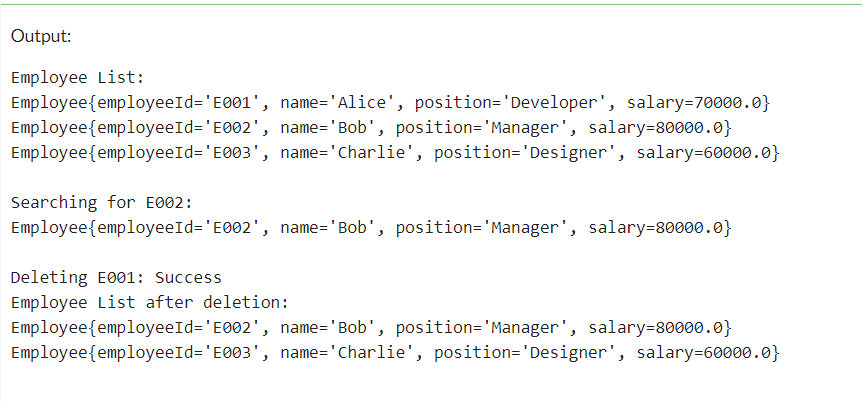
// Traversing employees after deletion

System.out.println("Employee List after deletion:");

management.traverseEmployees();

}

}



**Exercise 5: Task Management System**

**Code:**

class Task {

private String taskId;

private String taskName;

private String status;

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

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

}

public String getTaskId() {

return taskId;

}

public String getTaskName() {

return taskName;

}

public String getStatus() {

return status;

}

public String toString() {

return "Task{" +

"taskId='" + taskId + '\'' +

", taskName='" + taskName + '\'' +

", status='" + status + '\'' +

'}';

}

}

class TaskNode {

Task task;

TaskNode next;

public TaskNode(Task task) {

this.task = task;

this.next = null;

}

}

class TaskList {

private TaskNode head;

public TaskList() {

this.head = null;

}

// Method to add a task

public void addTask(Task task) {

TaskNode newNode = new TaskNode(task);

if (head == null) {

head = newNode; // If the list is empty, set the new node as head

} else {

TaskNode current = head;

while (current.next != null) {

current = current.next; // Traverse to the end of the list

}

current.next = newNode; // Add the new node at the end

}

}

// Method to search for a task by taskId

public Task searchTask(String taskId) {

TaskNode current = head;

while (current != null) {

if (current.task.getTaskId().equals(taskId)) {

return current.task; // Task found

}

current = current.next;

}

return null; // Task not found

}

// Method to traverse and display all tasks

public void traverseTasks() {

TaskNode current = head;

while (current != null) {

System.out.println(current.task);

current = current.next;

}

}

// Method to delete a task by taskId

public boolean deleteTask(String taskId) {

if (head == null) return false; // List is empty

// If the task to be deleted is the head

if (head.task.getTaskId().equals(taskId)) {

head = head.next; // Move head to the next node

return true; // Task deleted

}

TaskNode current = head;

while (current.next != null) {

if (current.next.task.getTaskId().equals(taskId)) {

current.next = current.next.next; // Bypass the node to delete it

return true; // Task deleted

}

current = current.next;

}

return false; // Task not found

}

}

class Main {

public static void main(String[] args) {

TaskList taskList = new TaskList();

// Adding tasks

taskList.addTask(new Task("T001", "Design UI", "In Progress"));

taskList.addTask(new Task("T002", "Implement Backend", "Pending"));

taskList.addTask(new Task("T003", "Testing", "Completed"));

// Traversing tasks

System.out.println("Task List:");

taskList.traverseTasks();

// Searching for a task

Task foundTask = taskList.searchTask("T002");

System.out.println("\nSearching for T002:");

System.out.println(foundTask != null ? foundTask : "Task not found");

// Deleting a task

boolean deleted = taskList.deleteTask("T001");

System.out.println("\nDeleting T001: " + (deleted ? "Success" : "Task not found"));

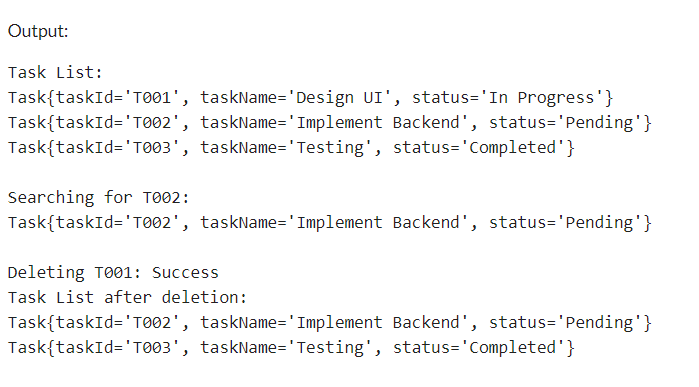
// Traversing tasks after deletion

System.out.println("Task List after deletion:");

taskList.traverseTasks();

}

}



**Exercise 6: Library Management System**

**Code:**

import java.util.Arrays;

class Book {

private String bookId;

private String title;

private String author;

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

this.bookId = bookId;

this.title = title;

this.author = author;

}

public String getBookId() {

return bookId;

}

public String getTitle() {

return title;

}

public String getAuthor() {

return author;

}

public String toString() {

return "Book{" +

"bookId='" + bookId + '\'' +

", title='" + title + '\'' +

", author='" + author + '\'' +

'}';

}

}

class BookSearch {

// Linear search implementation

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

for (Book book : books) {

if (book.getTitle().equalsIgnoreCase(targetTitle)) {

return book; // Book found

}

}

return null; // Book not found

}

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

int left = 0;

int right = books.length - 1;

while (left <= right) {

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

int comparison = books[mid].getTitle().compareToIgnoreCase(targetTitle);

if (comparison == 0) {

return books[mid]; // Book found

} else if (comparison < 0) {

left = mid + 1; // Search in the right half

} else {

right = mid - 1; // Search in the left half

}

}

return null; // Book not found

}

}

class Main {

public static void main(String[] args) {

Book[] books = {

new Book("B001", "The Great Gatsby", "F. Scott Fitzgerald"),

new Book("B002", "To Kill a Mockingbird", "Harper Lee"),

new Book("B003", "1984", "George Orwell"),

new Book("B004", "Pride and Prejudice", "Jane Austen"),

new Book("B005", "The Catcher in the Rye", "J.D. Salinger")

};

// Demonstrating Linear Search

String searchTitleLinear = "1984";

Book foundBookLinear = BookSearch.linearSearch(books, searchTitleLinear);

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

// Sorting books by title for Binary Search

Arrays.sort(books, (b1, b2) -> b1.getTitle().compareToIgnoreCase(b2.getTitle()));

// Demonstrating Binary Search

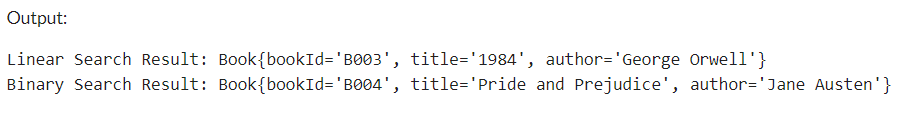
String searchTitleBinary = "Pride and Prejudice";

Book foundBookBinary = BookSearch.binarySearch(books, searchTitleBinary);

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

}

}



**Exercise 7: Financial Forecasting**

**Code:**

class FutureValueCalculator {

public static double calculateFutureValue(double initialValue, double growthRate, int periods) {

if (periods == 0) {

return initialValue; // Base case: return the initial value

} else {

double futureValue = calculateFutureValue(initialValue, growthRate, periods - 1);

return futureValue \* (1 + growthRate); // Recursive case: calculate future value

}

}

}

class Main {

public static void main(String[] args) {

double initialValue = 1000;

double growthRate = 0.05; // 5% growth rate

int periods = 5;

double futureValue = FutureValueCalculator.calculateFutureValue(initialValue, growthRate, periods);

System.out.println("Future Value: $" + futureValue);

}

}

