Solutions: Data Structures and Algorithms - Java Code

# Exercise 1: Inventory Management System

import java.util.HashMap;  
  
class Product {  
 int productId;  
 String productName;  
 int quantity;  
 double price;  
  
 public Product(int id, String name, int qty, double price) {  
 this.productId = id;  
 this.productName = name;  
 this.quantity = qty;  
 this.price = price;  
 }  
}  
  
class Inventory {  
 HashMap<Integer, Product> products = new HashMap<>();  
  
 public void addProduct(Product p) {  
 products.put(p.productId, p);  
 }  
  
 public void updateProduct(Product p) {  
 products.put(p.productId, p);  
 }  
  
 public void deleteProduct(int id) {  
 products.remove(id);  
 }  
}

# Exercise 2: E-commerce Platform Search Function

import java.util.Arrays;  
import java.util.Comparator;  
  
class Product {  
 int productId;  
 String productName;  
 String category;  
  
 public Product(int id, String name, String category) {  
 this.productId = id;  
 this.productName = name;  
 this.category = category;  
 }  
}  
  
class SearchFunction {  
 public static Product linearSearch(Product[] products, String name) {  
 for (Product p : products) {  
 if (p.productName.equals(name)) return p;  
 }  
 return null;  
 }  
  
 public static Product binarySearch(Product[] products, String name) {  
 Arrays.sort(products, Comparator.comparing(p -> p.productName));  
 int low = 0, high = products.length - 1;  
 while (low <= high) {  
 int mid = (low + high) / 2;  
 int cmp = products[mid].productName.compareTo(name);  
 if (cmp == 0) return products[mid];  
 else if (cmp < 0) low = mid + 1;  
 else high = mid - 1;  
 }  
 return null;  
 }  
}

# Exercise 3: Sorting Customer Orders

class Order {  
 int orderId;  
 String customerName;  
 double totalPrice;  
  
 public Order(int id, String name, double price) {  
 this.orderId = id;  
 this.customerName = name;  
 this.totalPrice = price;  
 }  
}  
  
// Bubble Sort  
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;  
 }  
 }  
 }  
}  
  
// Quick Sort  
void quickSort(Order[] arr, int low, int high) {  
 if (low < high) {  
 int pi = partition(arr, low, high);  
 quickSort(arr, low, pi - 1);  
 quickSort(arr, pi + 1, high);  
 }  
}  
  
int partition(Order[] arr, int low, int high) {  
 double pivot = arr[high].totalPrice;  
 int i = (low - 1);  
 for (int j = low; j < high; j++) {  
 if (arr[j].totalPrice <= pivot) {  
 i++;  
 Order temp = arr[i];  
 arr[i] = arr[j];  
 arr[j] = temp;  
 }  
 }  
 Order temp = arr[i + 1];  
 arr[i + 1] = arr[high];  
 arr[high] = temp;  
 return i + 1;  
}

# Exercise 4: Employee Management System

class Employee {  
 int employeeId;  
 String name;  
 String position;  
 double salary;  
  
 public Employee(int id, String name, String pos, double sal) {  
 this.employeeId = id;  
 this.name = name;  
 this.position = pos;  
 this.salary = sal;  
 }  
}  
  
class EmployeeManagement {  
 Employee[] employees = new Employee[100];  
 int count = 0;  
  
 void add(Employee e) {  
 if (count < employees.length)  
 employees[count++] = e;  
 }  
  
 Employee search(int id) {  
 for (int i = 0; i < count; i++) {  
 if (employees[i].employeeId == id)  
 return employees[i];  
 }  
 return null;  
 }  
  
 void traverse() {  
 for (int i = 0; i < count; i++) {  
 System.out.println(employees[i].name);  
 }  
 }  
  
 void delete(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];  
 }  
 count--;  
 break;  
 }  
 }  
 }  
}

# 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;  
 }  
}  
  
class TaskList {  
 Task head = null;  
  
 void add(Task task) {  
 if (head == null) head = task;  
 else {  
 Task current = head;  
 while (current.next != null) current = current.next;  
 current.next = task;  
 }  
 }  
  
 Task search(int id) {  
 Task current = head;  
 while (current != null) {  
 if (current.taskId == id) return current;  
 current = current.next;  
 }  
 return null;  
 }  
  
 void traverse() {  
 Task current = head;  
 while (current != null) {  
 System.out.println(current.taskName);  
 current = current.next;  
 }  
 }  
  
 void delete(int id) {  
 if (head == null) return;  
 if (head.taskId == id) {  
 head = head.next;  
 return;  
 }  
 Task current = head;  
 while (current.next != null) {  
 if (current.next.taskId == id) {  
 current.next = current.next.next;  
 return;  
 }  
 current = current.next;  
 }  
 }  
}

# Exercise 6: Library Management System

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;  
 }  
}  
  
class Library {  
 Book[] books;  
  
 public Book linearSearch(String title) {  
 for (Book book : books) {  
 if (book.title.equals(title)) return book;  
 }  
 return null;  
 }  
  
 public Book binarySearch(String title) {  
 Arrays.sort(books, Comparator.comparing(b -> b.title));  
 int low = 0, high = books.length - 1;  
 while (low <= high) {  
 int mid = (low + high) / 2;  
 int cmp = books[mid].title.compareTo(title);  
 if (cmp == 0) return books[mid];  
 else if (cmp < 0) low = mid + 1;  
 else high = mid - 1;  
 }  
 return null;  
 }  
}

# Exercise 7: Financial Forecasting

class Forecast {  
  
 double predictValue(int year, double[] growthRates) {  
 if (year == 0) return 1000; // Base value  
 return predictValue(year - 1, growthRates) \* (1 + growthRates[year - 1]);  
 }  
  
 double predictValueMemo(int year, double[] growthRates, double[] memo) {  
 if (year == 0) return 1000;  
 if (memo[year] != 0) return memo[year];  
 memo[year] = predictValueMemo(year - 1, growthRates, memo) \* (1 + growthRates[year - 1]);  
 return memo[year];  
 }  
}