## Exercise 1: Inventory Management System

Code to manage adding, updating, and deleting inventory products using HashMap. Shows product information and performs operations.

package com.inventory.example;

import java.util.HashMap;

import java.util.Scanner;

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 "Product ID: " + productId +

", Name: " + productName +

", Quantity: " + quantity +

", Price: $" + price;

}

}

public class InventoryManager {

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

public void addProduct(Product product) {

inventory.put(product.productId, product);

System.out.println("Product added.");

}

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

if (inventory.containsKey(productId)) {

Product product = inventory.get(productId);

product.productName = name;

product.quantity = quantity;

product.price = price;

System.out.println("Product updated.");

} else {

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

}

}

public void deleteProduct(int productId) {

if (inventory.remove(productId) != null) {

System.out.println("Product deleted.");

} else {

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

}

}

public void viewInventory() {

if (inventory.isEmpty()) {

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

} else {

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

System.out.println(p);

}

}

}

public static void main(String[] args) {

InventoryManager manager = new InventoryManager();

manager.addProduct(new Product(1, "Keyboard", 50, 799.99));

manager.addProduct(new Product(2, "Mouse", 100, 499.99));

manager.updateProduct(1, "Mechanical Keyboard", 60, 999.99);

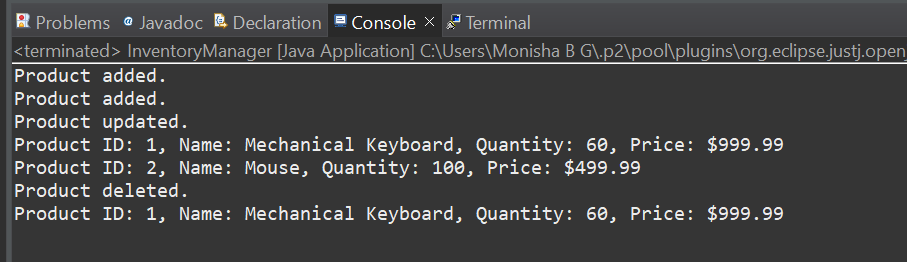
manager.viewInventory();

manager.deleteProduct(2);

manager.viewInventory();

}

}



## Exercise 2: E-commerce Platform Search Function

Implements linear and binary search for products by name. Shows how binary search requires a sorted array.

package com.ecommerce.example;

import java.util.Arrays;

import java.util.Comparator;

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;

}

public String toString() {

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

}

}

public class ECommerceSearch {

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

for (Product p : products) {

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

return p;

}

}

return null;

}

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

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

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

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

else if (cmp < 0) right = mid - 1;

else left = mid + 1;

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product(101, "Laptop", "Electronics"),

new Product(102, "Shoes", "Fashion"),

new Product(103, "Book", "Education"),

new Product(104, "Mobile", "Electronics"),

new Product(105, "Watch", "Accessories")

};

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

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

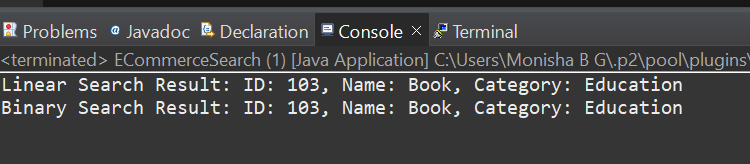
System.out.println("Linear Search Result: " + (result1 != null ? result1 : "Product not found"));

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

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

}

}



## Exercise 3: Sorting Customer Orders

Code to sort customer orders using Bubble Sort and Quick Sort by total price. Shows sorted outputs for comparison.

package com.order.search;

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 "Order ID: " + orderId + ", Customer: " + customerName + ", Total Price: $" + totalPrice;

}

}

public class OrderSorting {

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);

}

}

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 printOrders(Order[] orders) {

for (Order o : orders) {

System.out.println(o);

}

}

public static void main(String[] args) {

Order[] orders = {

new Order(101, "Alice", 250.0),

new Order(102, "Bob", 1200.0),

new Order(103, "Charlie", 500.0),

new Order(104, "Diana", 800.0),

new Order(105, "Eve", 200.0)

};

System.out.println("Original Orders:");

printOrders(orders);

Order[] bubbleSortedOrders = orders.clone();

bubbleSort(bubbleSortedOrders);

System.out.println("\nBubble Sorted Orders (by totalPrice):");

printOrders(bubbleSortedOrders);

Order[] quickSortedOrders = orders.clone();

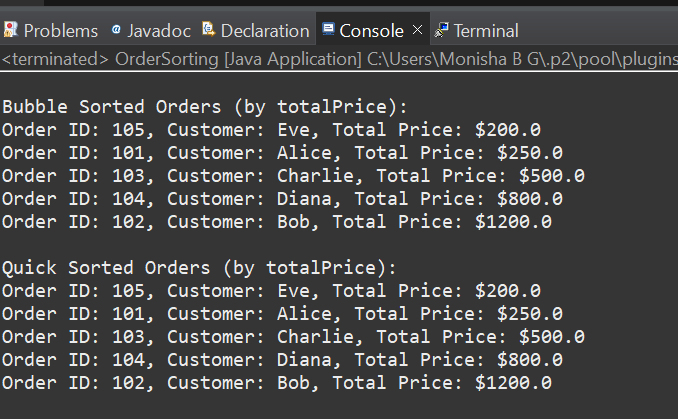
quickSort(quickSortedOrders, 0, quickSortedOrders.length - 1);

System.out.println("\nQuick Sorted Orders (by totalPrice):");

printOrders(quickSortedOrders);

}

}



## Exercise 4: Employee Management System

Manages employee records using an array. Supports adding, searching, deleting and traversing employees.

package com.employee.management;

class Employee {

int employeeId;

String name;

String position;

double salary;

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

this.employeeId = employeeId;

this.name = name;

this.position = position;

this.salary = salary;

}

public String toString() {

return "ID: " + employeeId + ", Name: " + name + ", Position: " + position + ", Salary: $" + salary;

}

}

public class EmployeeManagement {

static final int MAX\_EMPLOYEES = 100;

Employee[] employees = new Employee[MAX\_EMPLOYEES];

int count = 0;

public void addEmployee(Employee emp) {

if (count < MAX\_EMPLOYEES) {

employees[count++] = emp;

System.out.println("Employee added.");

} else {

System.out.println("Employee array is full!");

}

}

public Employee searchEmployee(int empId) {

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

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

return employees[i];

}

}

return null;

}

public void traverseEmployees() {

if (count == 0) {

System.out.println("No employees to display.");

return;

}

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

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

}

}

public void deleteEmployee(int empId) {

boolean found = false;

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

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

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

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

}

employees[--count] = null;

found = true;

System.out.println("Employee deleted.");

break;

}

}

if (!found) {

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

}

}

public static void main(String[] args) {

EmployeeManagement ems = new EmployeeManagement();

ems.addEmployee(new Employee(101, "Alice", "Manager", 80000));

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

ems.addEmployee(new Employee(103, "Charlie", "HR", 50000));

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

ems.traverseEmployees();

System.out.println("\nSearch Employee with ID 102:");

Employee result = ems.searchEmployee(102);

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

System.out.println("\nDeleting Employee with ID 101:");

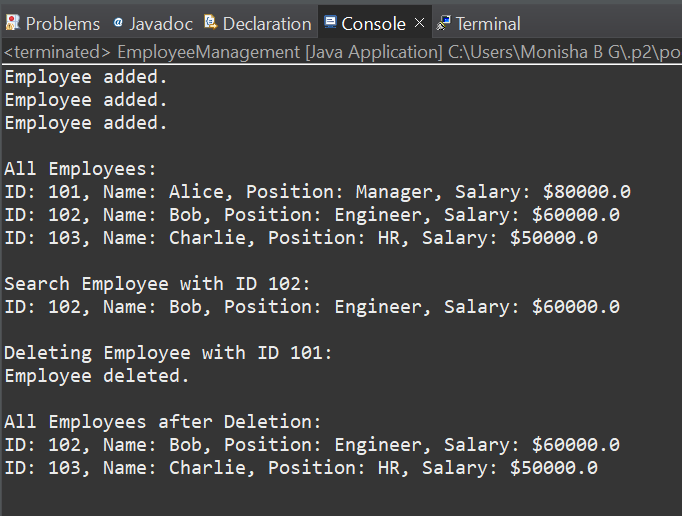
ems.deleteEmployee(101);

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

ems.traverseEmployees();

}

}



## Exercise 5: Task Management System

Implements a singly linked list to manage tasks. Supports adding, searching, deleting, and traversing tasks.

package com.task.manager;

class Task {

int taskId;

String taskName;

String status;

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

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

}

public String toString() {

return "Task ID: " + taskId + ", Name: " + taskName + ", Status: " + status;

}

}

class Node {

Task task;

Node next;

public Node(Task task) {

this.task = task;

this.next = null;

}

}

public class TaskManager {

Node head;

public void addTask(Task task) {

Node newNode = new Node(task);

if (head == null) {

head = newNode;

} else {

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

System.out.println("Task added.");

}

public Task searchTask(int taskId) {

Node current = head;

while (current != null) {

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

return current.task;

}

current = current.next;

}

return null;

}

public void traverseTasks() {

if (head == null) {

System.out.println("No tasks to display.");

return;

}

Node current = head;

while (current != null) {

System.out.println(current.task);

current = current.next;

}

}

// Delete task by ID

public void deleteTask(int taskId) {

if (head == null) {

System.out.println("Task list is empty.");

return;

}

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

head = head.next;

System.out.println("Task deleted.");

return;

}

Node current = head;

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

current = current.next;

}

if (current.next == null) {

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

} else {

current.next = current.next.next;

System.out.println("Task deleted.");

}

}

public static void main(String[] args) {

TaskManager manager = new TaskManager();

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

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

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

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

manager.traverseTasks();

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

Task found = manager.searchTask(2);

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

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

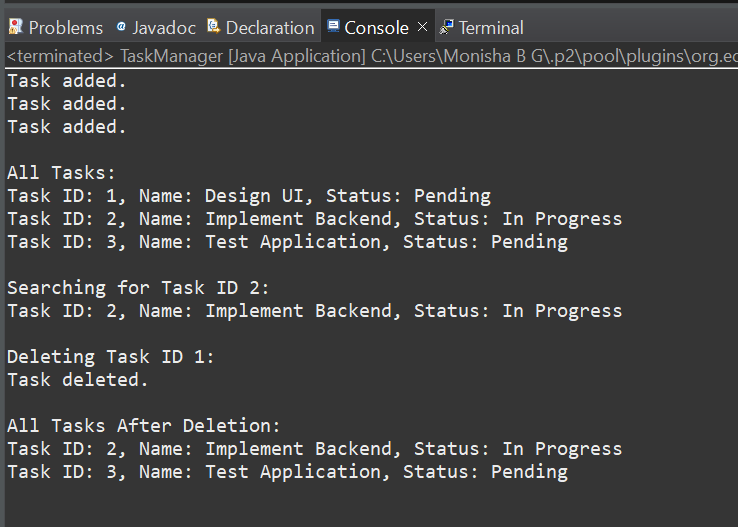
manager.deleteTask(1);

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

manager.traverseTasks();

}

}



## Exercise 6: Library Management System

Performs linear and binary search on books by title. Books are sorted before binary search.

package com.library.system;

import java.util.Arrays;

import java.util.Comparator;

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;

}

public String toString() {

return "Book ID: " + bookId + ", Title: " + title + ", Author: " + author;

}

}

public class LibrarySystem {

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

for (Book book : books) {

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

return book;

}

}

return null;

}

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

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

while (left <= right) {

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

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

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

else if (cmp < 0) right = mid - 1;

else left = mid + 1;

}

return null;

}

public static void printBooks(Book[] books) {

for (Book book : books) {

System.out.println(book);

}

}

public static void main(String[] args) {

Book[] books = {

new Book(101, "Data Structures", "Mark Allen"),

new Book(102, "Operating Systems", "Andrew Tanenbaum"),

new Book(103, "Introduction to Algorithms", "Thomas Cormen"),

new Book(104, "Database Systems", "Raghu Ramakrishnan"),

new Book(105, "Computer Networks", "Andrew Tanenbaum")

};

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

System.out.println("Books in Library:");

printBooks(books);

System.out.println("\nLinear Search for 'Operating Systems':");

Book result1 = linearSearch(books, "Operating Systems");

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

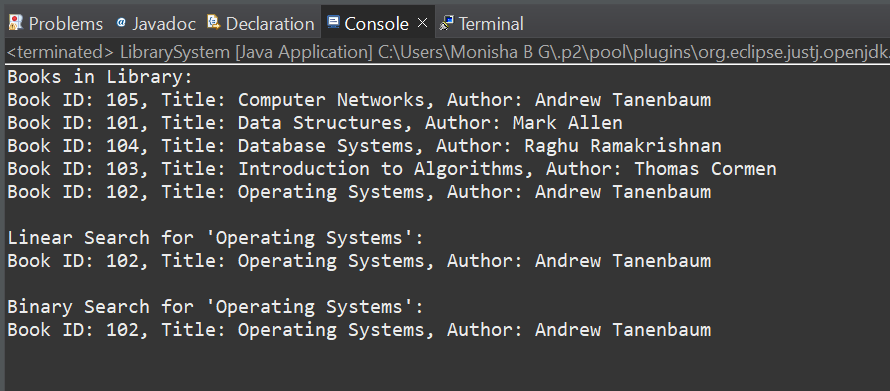
System.out.println("\nBinary Search for 'Operating Systems':");

Book result2 = binarySearch(books, "Operating Systems");

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

}

}



## Exercise 7: Financial Forecasting

Recursive calculation of future value with and without memoization. Forecasts 5 years of financial growth.

package com.financial.forecasting;

public class FinancialForecasting {

public static double calculateFutureValue(double initialAmount, double rate, int year) {

if (year == 0) {

return initialAmount;

}

return calculateFutureValue(initialAmount, rate, year - 1) \* (1 + rate);

}

public static double calculateFutureValueMemo(double initialAmount, double rate, int year, double[] memo) {

if (year == 0) return initialAmount;

if (memo[year] != 0) return memo[year];

memo[year] = calculateFutureValueMemo(initialAmount, rate, year - 1, memo) \* (1 + rate);

return memo[year];

}

public static void main(String[] args) {

double initialAmount = 10000;

double growthRate = 0.08;

int forecastYears = 5;

System.out.println(" Recursive Forecast (No Memoization):");

for (int year = 0; year <= forecastYears; year++) {

double futureValue = calculateFutureValue(initialAmount, growthRate, year);

System.out.printf("Year %d: ₹%.2f\n", year, futureValue);

}

System.out.println("\n Optimized Recursive Forecast (Memoization):");

double[] memo = new double[forecastYears + 1];

for (int year = 0; year <= forecastYears; year++) {

double futureValue = calculateFutureValueMemo(initialAmount, growthRate, year, memo);

System.out.printf("Year %d: ₹%.2f\n", year, futureValue);

}

}

}

