**Exercise 1: Inventory Management System**

Code

import java.util.\*;

class Product {

    String productId;

    String productName;

    int quantity;

    double price;

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

        this.productId = productId;

        this.productName = productName;

        this.quantity = quantity;

        this.price = price;

    }

    @Override

    public String toString() {

        return "ProductID: " + productId + ", Name: " + productName + ", Quantity: " + quantity + ", Price: " + price;

    }

}

public class InventoryManagementSystem {

    private static Map<String, Product> inventory = new HashMap<>();

    // Add product

    public static void addProduct(Product product) {

        inventory.put(product.productId, product);

        System.out.println("Added: " + product);

    }

    // Update product

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

        if (inventory.containsKey(productId)) {

            Product p = inventory.get(productId);

            p.quantity = quantity;

            p.price = price;

            System.out.println("Updated: " + p);

        } else {

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

        }

    }

    // Delete product

    public static void deleteProduct(String productId) {

        if (inventory.containsKey(productId)) {

            Product removed = inventory.remove(productId);

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

        } else {

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

        }

    }

    // Display all products

    public static void displayInventory() {

        System.out.println("\nCurrent Inventory:");

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

            System.out.println(p);

        }

    }

    public static void main(String[] args) {

        addProduct(new Product("P001", "Laptop", 10, 55000));

        addProduct(new Product("P002", "Mouse", 100, 500));

        updateProduct("P002", 90, 450);

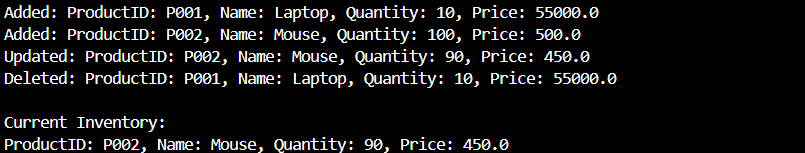
        deleteProduct("P001");

        displayInventory();

    }

}

Output



**Exercise 2: E-commerce Platform Search Function (Mandatory hands-on)**

Code

class Product {

    String productId;

    String productName;

    String category;

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

        this.productId = productId;

        this.productName = productName;

        this.category = category;

    }

    @Override

    public String toString() {

        return "[" + productId + "] " + productName + " (" + category + ")";

    }

}

import java.util.\*;

public class EcommerceSearch {

    // Linear Search by product name

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

        for (Product p : products) {

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

                return p;

            }

        }

        return null;

    }

    // Binary Search by product name

    public static Product binarySearch(Product[] 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;

    }

    // Sort products by name

    public static void sortProducts(Product[] products) {

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

    }

    public static void main(String[] args) {

        Product[] products = {

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

            new Product("P002", "Shoes", "Fashion"),

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

            new Product("P004", "Book", "Education"),

            new Product("P005", "Watch", "Accessories")

        };

        // Linear Search

        String searchName = "Camera";

        Product result1 = linearSearch(products, searchName);

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

        // Binary Search )

        sortProducts(products);  // Important step before binary search

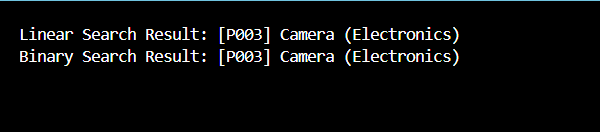
        Product result2 = binarySearch(products, searchName);

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

    }

}

Output



**Exercise 3: Sorting Customer Orders**

Code

class Order {

    String orderId;

    String customerName;

    double totalPrice;

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

        this.orderId = orderId;

        this.customerName = customerName;

        this.totalPrice = totalPrice;

    }

    @Override

    public String toString() {

        return "[" + orderId + "] " + customerName + " - " + totalPrice;

    }

}

public class OrderSorting {

    // Bubble Sort (Ascending by totalPrice)

    public static void bubbleSort(Order[] orders) {

        int n = orders.length;

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

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

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

                    Order temp = orders[j];

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

                    orders[j + 1] = temp;

                }

            }

        }

    }

    // Quick Sort (Descending by totalPrice)

    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) { // descending

                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;

    }

    // Utility to print orders

    public static void printOrders(Order[] orders) {

        for (Order order : orders) {

            System.out.println(order);

        }

        System.out.println();

    }

    public static void main(String[] args) {

        Order[] orders = {

            new Order("O001", "Alice", 2500),

            new Order("O002", "Bob", 5600),

            new Order("O003", "Charlie", 1200),

            new Order("O004", "Diana", 8000),

            new Order("O005", "Ethan", 3400)

        };

        // Bubble Sort

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

        bubbleSort(orders);

        printOrders(orders);

        // Quick Sort

        System.out.println("Quick Sort (Descending by totalPrice):");

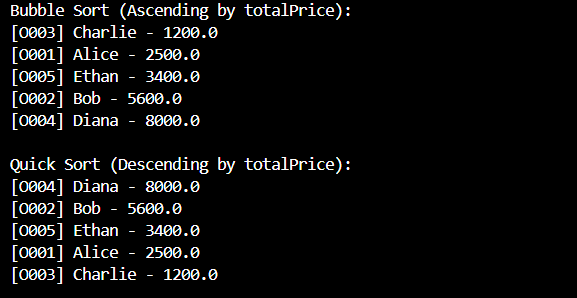
        quickSort(orders, 0, orders.length - 1);

        printOrders(orders);

    }

}

Output



**Exercise 4: Employee Management System**

Code

class Employee {

    String employeeId;

    String name;

    String position;

    double salary;

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

        this.employeeId = employeeId;

        this.name = name;

        this.position = position;

        this.salary = salary;

    }

    @Override

    public String toString() {

        return "[" + employeeId + "] " + name + " - " + position + " - " + salary;

    }

}

public class EmployeeManagementSystem {

    static final int MAX\_EMPLOYEES = 100;

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

    static int count = 0;

    // Add employee

    public static void addEmployee(Employee e) {

        if (count < MAX\_EMPLOYEES) {

            employees[count++] = e;

            System.out.println("Employee added: " + e);

        } else {

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

        }

    }

    // Search by employeeId

    public static Employee searchEmployee(String empId) {

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

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

                return employees[i];

            }

        }

        return null;

    }

    // Traverse all employees

    public static void traverseEmployees() {

        System.out.println("\nEmployee List:");

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

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

        }

    }

    // Delete employee by ID

    public static void deleteEmployee(String empId) {

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

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

                // Shift elements left

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

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

                }

                employees[--count] = null;

                System.out.println("Employee deleted: " + empId);

                return;

            }

        }

        System.out.println("Employee not found: " + empId);

    }

    public static void main(String[] args) {

        // Sample operations

        addEmployee(new Employee("E001", "Alice", "Manager", 75000));

        addEmployee(new Employee("E002", "Bob", "Developer", 50000));

        addEmployee(new Employee("E003", "Charlie", "Designer", 45000));

        traverseEmployees();

        System.out.println("\nSearch Result:");

        Employee e = searchEmployee("E002");

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

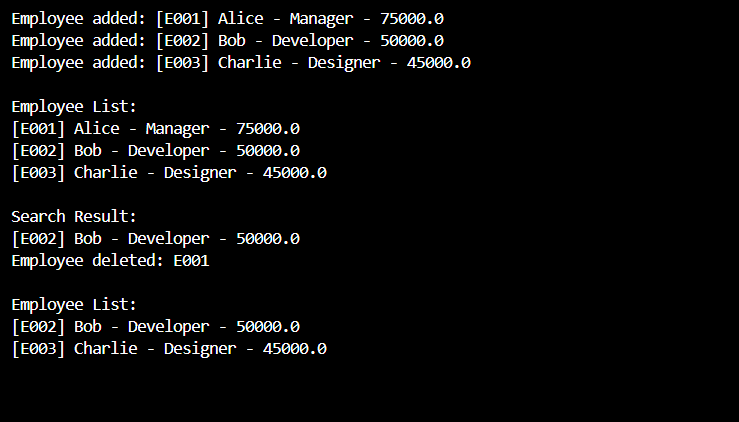
        deleteEmployee("E001");

        traverseEmployees();

    }

}

Output



**Exercise 5: Task Management System**

Code

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;

    }

    @Override

    public String toString() {

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

    }

}

public class TaskNode {

    Task task;

    TaskNode next;

    public TaskNode(Task task) {

        this.task = task;

        this.next = null;

    }

}

public class TaskLinkedList {

    private TaskNode head;

    public void addTask(Task task) {

        TaskNode newNode = new TaskNode(task);

        if (head == null) {

            head = newNode;

        } else {

            TaskNode current = head;

            while (current.next != null) {

                current = current.next;

            }

            current.next = newNode;

        }

    }

    public Task searchTask(int taskId) {

        TaskNode current = head;

        while (current != null) {

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

                return current.task;

            }

            current = current.next;

        }

        return null;

    }

    public void deleteTask(int taskId) {

        if (head == null) return;

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

            head = head.next;

            return;

        }

        TaskNode current = head;

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

            current = current.next;

        }

        if (current.next != null) {

            current.next = current.next.next;

        }

    }

    public void displayTasks() {

        TaskNode current = head;

        while (current != null) {

            System.out.println(current.task);

            current = current.next;

        }

    }

}

public class TaskManager {

    public static void main(String[] args) {

        TaskLinkedList taskList = new TaskLinkedList();

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

        taskList.addTask(new Task(2, "Write Backend", "In Progress"));

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

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

        taskList.displayTasks();

        System.out.println("\nSearch Task with ID 2:");

        Task t = taskList.searchTask(2);

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

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

        taskList.deleteTask(1);

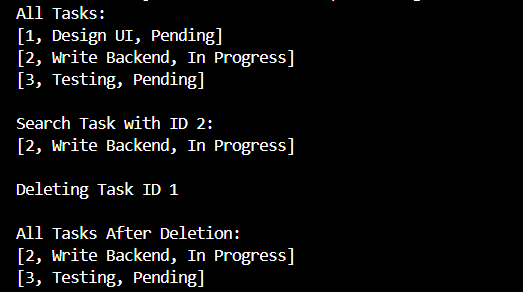
        System.out.println("\nAll Tasks After Deletion:");

        taskList.displayTasks();

    }

}

Output



**Exercise 6: Library Management System**

Code

public 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 + ", " + title + ", " + author + "]";

    }

}

import java.util.Arrays;

import java.util.Comparator;

public class Library {

    // Linear Search

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

        for (Book book : books) {

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

                return book;

            }

        }

        return null;

    }

    // Binary Search

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

        int low = 0;

        int high = books.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

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

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

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

            else high = mid - 1;

        }

        return null;

    }

    public static void main(String[] args) {

        Book[] books = {

            new Book(101, "Data Structures", "Narasimha Karumanchi"),

            new Book(102, "Java Programming", "Herbert Schildt"),

            new Book(103, "Algorithms", "Robert Sedgewick"),

            new Book(104, "Python Basics", "Guido van Rossum"),

            new Book(105, "Machine Learning", "Tom Mitchell")

        };

        System.out.println("Linear Search: Searching for 'Python Basics'");

        Book result1 = linearSearch(books, "Python Basics");

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

        // Sort books by title for binary search

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

        System.out.println("\n Binary Search: Searching for 'Java Programming'");

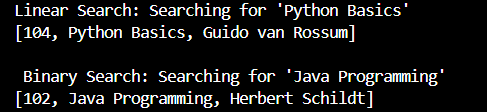
        Book result2 = binarySearch(books, "Java Programming");

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

    }

}

Output



**Exercise 7: Financial Forecasting (Mandatory hands-on)**

Code

public class FinancialForecast {

    // Recursive method

    public static double forecastRecursive(double initialValue, double rate, int years) {

        if (years == 0) {

            return initialValue;

        }

        return forecastRecursive(initialValue, rate, years - 1) \* (1 + rate);

    }

    public static double forecastMemoized(double initialValue, double rate, int years) {

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

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

            memo[i] = -1;

        }

        return helper(initialValue, rate, years, memo);

    }

    private static double helper(double initialValue, double rate, int year, double[] memo) {

        if (year == 0) return initialValue;

        if (memo[year] != -1) return memo[year];

        memo[year] = helper(initialValue, rate, year - 1, memo) \* (1 + rate);

        return memo[year];

    }

    // Iterative method

    public static double forecastIterative(double initialValue, double rate, int years) {

        double value = initialValue;

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

            value \*= (1 + rate);

        }

        return value;

    }

    public static void main(String[] args) {

        double initialValue = 10000.0;

        double growthRate = 0.05; // 5% growth

        int years = 5;

        double recursiveResult = forecastRecursive(initialValue, growthRate, years);

        double memoizedResult = forecastMemoized(initialValue, growthRate, years);

        double iterativeResult = forecastIterative(initialValue, growthRate, years);

        System.out.printf("Future Value (Recursive)  : %.2f%n", recursiveResult);

        System.out.printf("Future Value (Memoized)   : %.2f%n", memoizedResult);

        System.out.printf("Future Value (Iterative)  : %.2f%n", iterativeResult);

    }

}

Output

