**WEEK-1**

**Exercise 1: Inventory Management System**

**InventoryManagementSystem.java:**

import java.util.HashMap;

import java.util.Scanner;

// Product class

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 "ID: " + productId + ", Name: " + productName +

               ", Quantity: " + quantity + ", Price: ₹" + price;

    }

}

// Main Inventory Manager

public class InventoryManagementSystem {

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

    static Scanner sc = new Scanner(System.in);

    public static void main(String[] args) {

        int choice;

        do {

            System.out.println("\n--- Inventory Management System ---");

            System.out.println("1. Add Product");

            System.out.println("2. Update Product");

            System.out.println("3. Delete Product");

            System.out.println("4. View Inventory");

            System.out.println("5. Exit");

            System.out.print("Enter your choice: ");

            choice = sc.nextInt();

            switch (choice) {

                case 1 -> addProduct();

                case 2 -> updateProduct();

                case 3 -> deleteProduct();

                case 4 -> viewInventory();

                case 5 -> System.out.println("Exiting...");

                default -> System.out.println("Invalid choice!");

            }

        } while (choice != 5);

    }

    static void addProduct() {

        System.out.print("Enter Product ID: ");

        int id = sc.nextInt();

        sc.nextLine(); // clear newline

        System.out.print("Enter Product Name: ");

        String name = sc.nextLine();

        System.out.print("Enter Quantity: ");

        int qty = sc.nextInt();

        System.out.print("Enter Price: ");

        double price = sc.nextDouble();

        Product p = new Product(id, name, qty, price);

        inventory.put(id, p);

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

    }

    static void updateProduct() {

        System.out.print("Enter Product ID to update: ");

        int id = sc.nextInt();

        if (inventory.containsKey(id)) {

            System.out.print("Enter New Quantity: ");

            int qty = sc.nextInt();

            System.out.print("Enter New Price: ");

            double price = sc.nextDouble();

            Product p = inventory.get(id);

            p.quantity = qty;

            p.price = price;

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

        } else {

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

        }

    }

    static void deleteProduct() {

        System.out.print("Enter Product ID to delete: ");

        int id = sc.nextInt();

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

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

        } else {

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

        }

    }

    static void viewInventory() {

        if (inventory.isEmpty()) {

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

            return;

        }

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

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

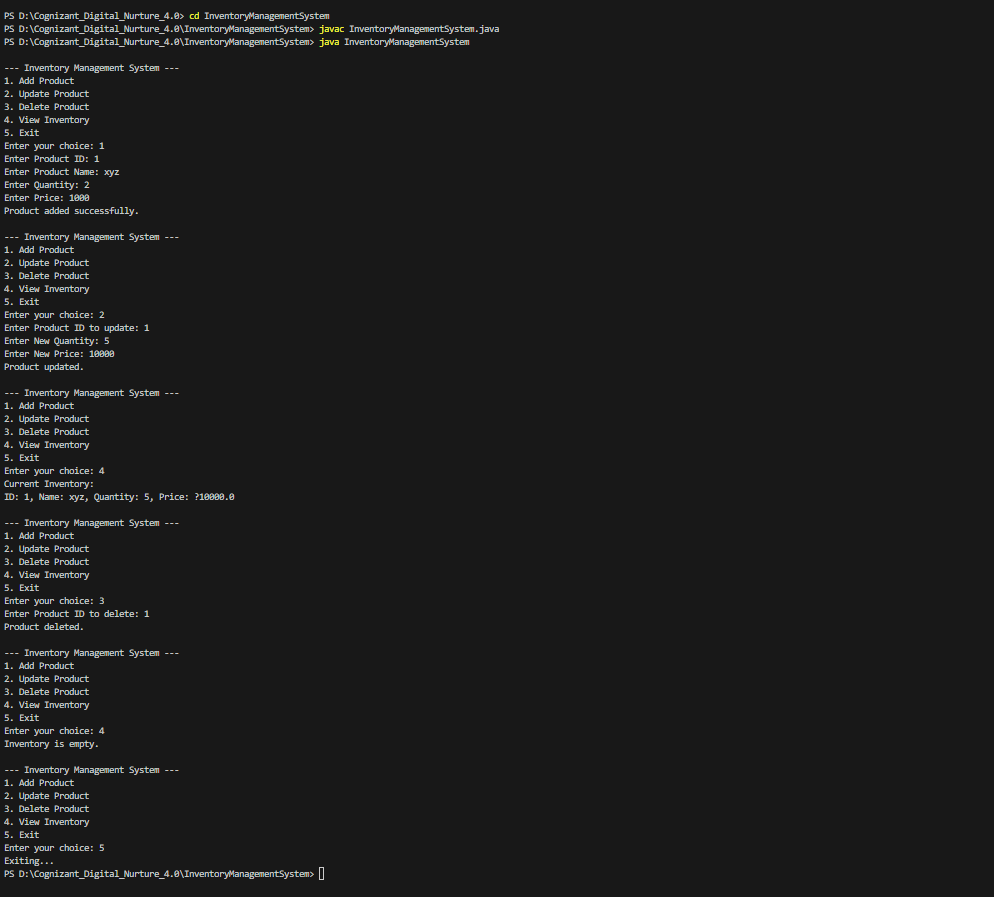
            System.out.println(p);

        }

}

}

**Output:**



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

**SearchDemo.java:**

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 "[" + productId + "] " + productName + " - " + category;

    }

}

public class SearchDemo {

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

        for (Product p : products) {

            if (p.productId == targetId) {

                return p;

            }

        }

        return null;

    }

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

        int low = 0;

        int high = products.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            if (products[mid].productId == targetId) {

                return products[mid];

            } else if (products[mid].productId < targetId) {

                low = mid + 1;

            } else {

                high = mid - 1;

            }

        }

        return null;

    }

    public static void main(String[] args) {

        Product[] products = {

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

            new Product(105, "Phone", "Electronics"),

            new Product(103, "T-shirt", "Fashion"),

            new Product(108, "Shoes", "Fashion"),

            new Product(102, "Book", "Education")

        };

        int targetId = 103;

        Product result1 = linearSearch(products, targetId);

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

        java.util.Arrays.sort(products, java.util.Comparator.comparingInt(p -> p.productId));

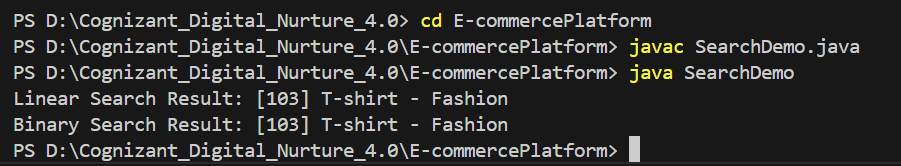
        Product result2 = binarySearch(products, targetId);

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

    }

}

**Output:**



**Exercise 3: Sorting Customer Orders**

**OrderSorting.java:**

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 "OrderID: " + orderId + ", Customer: " + customerName + ", Total Price: " + totalPrice;

    }

}

public class OrderSorting {

    // Bubble Sort

    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) {

                    Order temp = orders[j];

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

                    orders[j + 1] = temp;

                }

            }

        }

    }

    // Quick Sort

    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 main(String[] args) {

        Order[] orders = {

            new Order(101, "Alice", 550.50),

            new Order(102, "Bob", 1200.00),

            new Order(103, "Charlie", 300.75),

            new Order(104, "Daisy", 999.99)

        };

        // Clone original array for both sorting methods

        Order[] ordersForBubble = orders.clone();

        Order[] ordersForQuick = orders.clone();

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

        for (Order o : orders) {

            System.out.println(o);

        }

        // Bubble Sort

        bubbleSort(ordersForBubble);

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

        for (Order o : ordersForBubble) {

            System.out.println(o);

        }

        // Quick Sort

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

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

        for (Order o : ordersForQuick) {

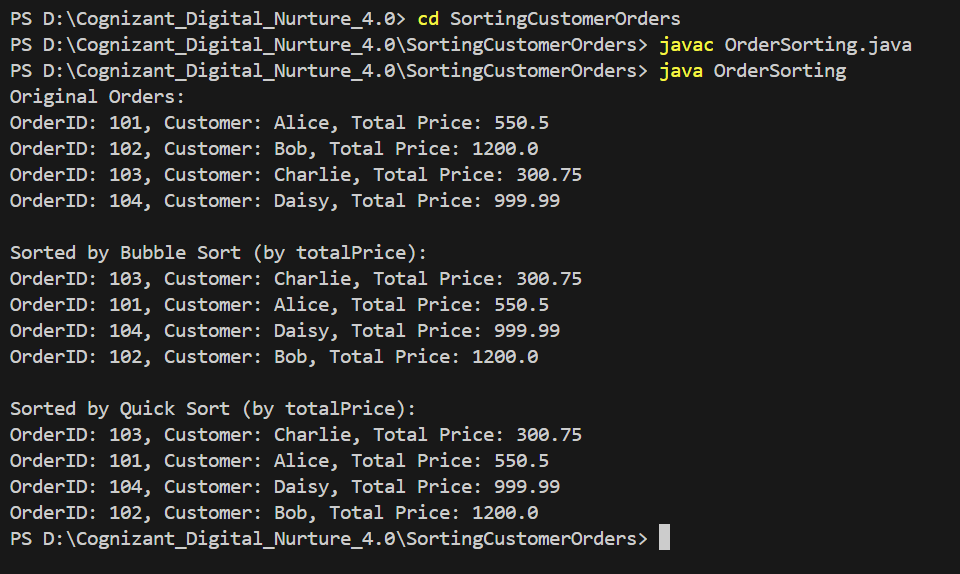
            System.out.println(o);

        }

    }

}

**Output:**



**Exercise 4: Employee Management System**

**EmployeeManagementSystem.java:**

import java.util.Scanner;

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 = name;

        this.position = position;

        this.salary = salary;

    }

    public void displayInfo() {

        System.out.println("ID: " + employeeId + ", Name: " + name + ", Position: " + position + ", Salary: ₹" + salary);

    }

}

public class EmployeeManagementSystem {

    static final int MAX\_EMPLOYEES = 100;

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

    static int count = 0;

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int choice;

        do {

            System.out.println("\n--- Employee Management System ---");

            System.out.println("1. Add Employee");

            System.out.println("2. Search Employee");

            System.out.println("3. View All Employees");

            System.out.println("4. Delete Employee");

            System.out.println("5. Exit");

            System.out.print("Enter your choice: ");

            choice = scanner.nextInt();

            scanner.nextLine();

            switch (choice) {

                case 1:

                    addEmployee(scanner);

                    break;

                case 2:

                    searchEmployee(scanner);

                    break;

                case 3:

                    traverseEmployees();

                    break;

                case 4:

                    deleteEmployee(scanner);

                    break;

                case 5:

                    System.out.println("Exiting... Thank you!");

                    break;

                default:

                    System.out.println("Invalid choice. Try again.");

            }

        } while (choice != 5);

    }

    public static void addEmployee(Scanner scanner) {

        if (count >= MAX\_EMPLOYEES) {

            System.out.println("Employee list full. Cannot add more.");

            return;

        }

        System.out.print("Enter ID: ");

        int id = scanner.nextInt();

        scanner.nextLine();

        System.out.print("Enter Name: ");

        String name = scanner.nextLine();

        System.out.print("Enter Position: ");

        String position = scanner.nextLine();

        System.out.print("Enter Salary: ");

        double salary = scanner.nextDouble();

        employees[count++] = new Employee(id, name, position, salary);

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

    }

    public static void searchEmployee(Scanner scanner) {

        System.out.print("Enter Employee ID to search: ");

        int id = scanner.nextInt();

        boolean found = false;

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

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

                employees[i].displayInfo();

                found = true;

                break;

            }

        }

        if (!found) {

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

        }

    }

public static void traverseEmployees() {

        if (count == 0) {

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

            return;

        }

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

            employees[i].displayInfo();

        }

    }

    public static void deleteEmployee(Scanner scanner) {

        System.out.print("Enter Employee ID to delete: ");

        int id = scanner.nextInt();

        boolean found = false;

        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];

                }

                employees[--count] = null;

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

                found = true;

                break;

            }

        }

        if (!found) {

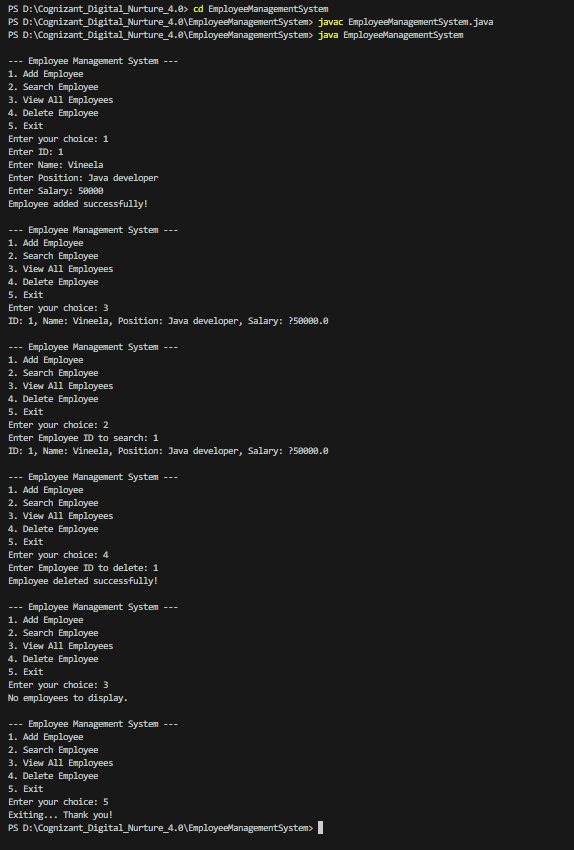
            System.out.println("Employee not found.");

        }

    }

}

**Output:**



**Exercise 5: Task Management System**

**TaskManagementSystem.java:**

class Task {

    int taskId;

    String taskName;

    String status;

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

        this.taskId = taskId;

        this.taskName = taskName;

        this.status = status;

    }

    public String toString() {

        return "TaskID: " + taskId + ", Name: " + taskName + ", Status: " + status;

    }

}

class TaskNode {

    Task task;

    TaskNode next;

    TaskNode(Task task) {

        this.task = task;

        this.next = null;

    }

}

class TaskLinkedList {

    TaskNode head;

    // Add Task

    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;

        }

        System.out.println("Task added: " + task.taskName);

    }

    // Traverse and Display All Tasks

    public void displayTasks() {

        if (head == null) {

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

            return;

        }

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

        TaskNode current = head;

        while (current != null) {

            System.out.println(current.task);

            current = current.next;

        }

    }

    // Search Task by ID

    public Task searchTask(int taskId) {

        TaskNode current = head;

        while (current != null) {

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

                return current.task;

            }

            current = current.next;

        }

        return null;

    }

    // 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 with ID " + taskId + " deleted.");

            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;

            System.out.println("Task with ID " + taskId + " deleted.");

        } else {

            System.out.println("Task with ID " + taskId + " not found.");

        }

    }

}

public class TaskManagementSystem {

    public static void main(String[] args) {

        TaskLinkedList taskList = new TaskLinkedList();

        // Adding tasks

        taskList.addTask(new Task(101, "Design UI", "Pending"));

        taskList.addTask(new Task(102, "Develop Backend", "In Progress"));

        taskList.addTask(new Task(103, "Testing", "Not Started"));

        // Display tasks

        taskList.displayTasks();

        // Search for a task

        Task foundTask = taskList.searchTask(102);

        if (foundTask != null) {

            System.out.println("Task Found: " + foundTask);

        } else {

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

        }

        // Delete a task

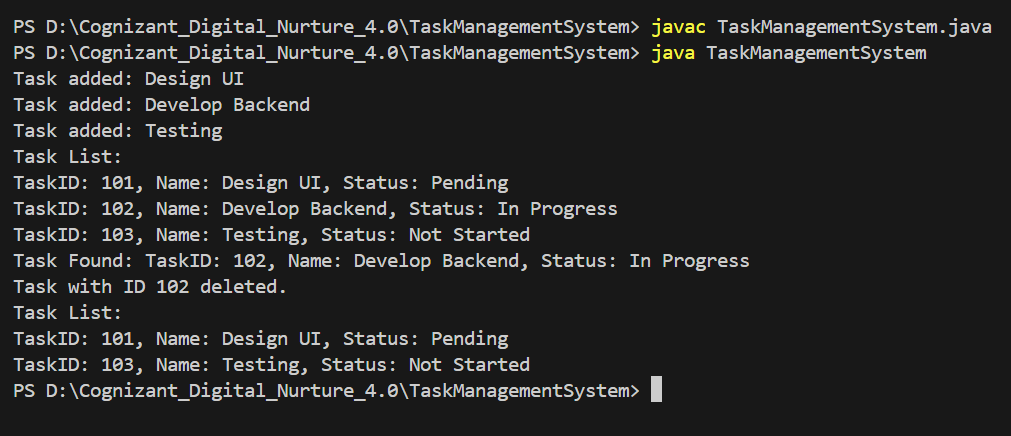
        taskList.deleteTask(102);

        taskList.displayTasks();

    }

}

**Output:**



**Exercise 6: Library Management System**

**LibrarySearch.java:**

import java.util.\*;

// Book class with bookId, title, and author

class Book {

    int bookId;

    String title;

    String author;

    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 LibrarySearch {

    // Linear Search

    public static List<Book> linearSearch(List<Book> books, String title) {

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

        for (Book book : books) {

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

                found.add(book);

            }

        }

        return found;

    }

    // Binary Search (Assumes sorted by title)

    public static Book binarySearch(List<Book> books, String title) {

        int low = 0, high = books.size() - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            Book midBook = books.get(mid);

            int cmp = midBook.title.compareToIgnoreCase(title);

            if (cmp == 0)

                return midBook;

            else if (cmp < 0)

                low = mid + 1;

            else

                high = mid - 1;

        }

        return null;

    }

    public static void main(String[] args) {

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

        books.add(new Book(101, "Data Structures", "Mark Allen"));

        books.add(new Book(102, "Java Programming", "James Gosling"));

        books.add(new Book(103, "Algorithms", "Robert Sedgewick"));

        books.add(new Book(104, "Python Basics", "Guido van Rossum"));

        books.add(new Book(105, "C++ Guide", "Bjarne Stroustrup"));

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter book title to search: ");

        String searchTitle = sc.nextLine();

        // Linear Search

        List<Book> resultLinear = linearSearch(books, searchTitle);

        if (!resultLinear.isEmpty()) {

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

            for (Book b : resultLinear) {

                System.out.println(b);

            }

        } else {

            System.out.println("No book found using Linear Search.");

        }

        // Sort list by title before binary search

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

        // Binary Search

        Book resultBinary = binarySearch(books, searchTitle);

        if (resultBinary != null) {

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

            System.out.println(resultBinary);

        } else {

            System.out.println("No book found using Binary Search.");

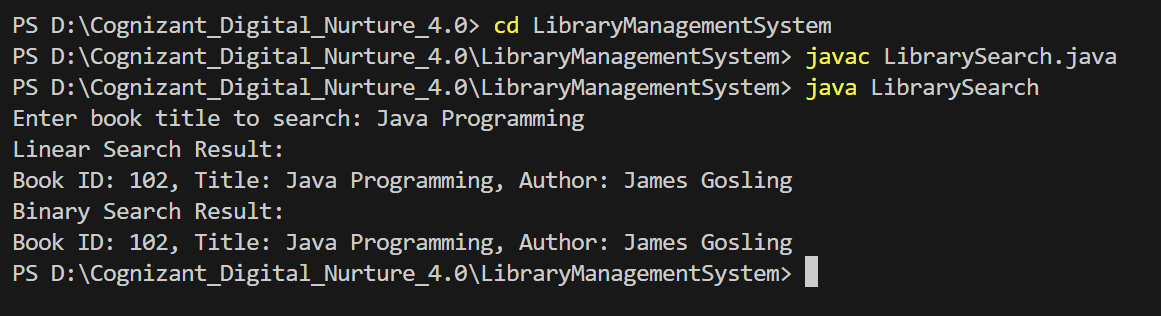
        }

        sc.close();

    }

}

**Output:**



**Exercise 7: Financial Forecasting**

**FinancialForecast.java:**

public class FinancialForecast {

    // Recursive method to calculate future value

    public static double calculateFutureValue(double presentValue, double growthRate, int years) {

        if (years == 0) {

            return presentValue;

        }

        // Recursive step: reduce years and apply growth rate

        return calculateFutureValue(presentValue, growthRate, years - 1) \* (1 + growthRate);

    }

    public static void main(String[] args) {

        double presentValue = 10000; // Initial investment

        double growthRate = 0.05;    // 5% annual growth

        int years = 5;

        double futureValue = calculateFutureValue(presentValue, growthRate, years);

        System.out.printf("Future Value after %d years: %.2f\n", years, futureValue);

    }

}

**Output:**

