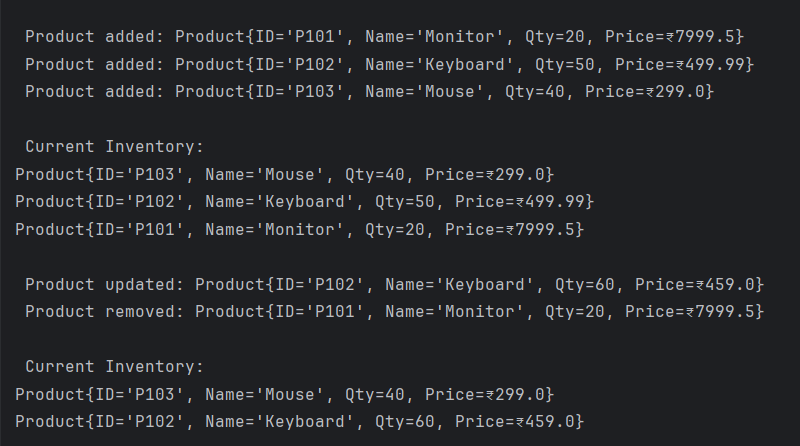
**Exercise 1: Inventory Management System**

**Code :**

import java.util.\*;  
  
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;  
 }  
  
 public String getProductId() { return productId; }  
 public void setQuantity(int quantity) { this.quantity = quantity; }  
 public void setPrice(double price) { this.price = price; }  
  
 @Override  
 public String toString() {  
 return "Product{" + "ID='" + productId + "', Name='" + productName +  
 "', Qty=" + quantity + ", Price=₹" + price + '}';  
 }  
}  
  
class InventoryManager {  
 private Map<String, Product> inventory = new HashMap<>();  
  
 public void addProduct(Product product) {  
 inventory.put(product.getProductId(), product);  
 System.*out*.println(" Product added: " + product);  
 }  
  
 public void updateProduct(String productId, int newQuantity, double newPrice) {  
 Product p = inventory.get(productId);  
 if (p != null) {  
 p.setQuantity(newQuantity);  
 p.setPrice(newPrice);  
 System.*out*.println(" Product updated: " + p);  
 } else {  
 System.*out*.println(" Product not found: " + productId);  
 }  
 }  
  
 public void deleteProduct(String productId) {  
 Product removed = inventory.remove(productId);  
 if (removed != null) {  
 System.*out*.println(" Product removed: " + removed);  
 } else {  
 System.*out*.println(" Product not found: " + productId);  
 }  
 }  
  
 public void displayInventory() {  
  
 if (inventory.isEmpty()) {  
 System.*out*.println(" Inventory is empty.");  
 } else {  
 System.*out*.println(" Current Inventory:");  
 for (Product p : inventory.values()) {  
 System.*out*.println(p);  
 }  
 }  
 }  
}  
  
public class InventoryManagementSystem {  
 public static void main(String[] args) {  
 InventoryManager manager = new InventoryManager();  
 System.*out*.println("\n");  
 Product p1 = new Product("P101", "Monitor", 20, 7999.50);  
 Product p2 = new Product("P102", "Keyboard", 50, 499.99);  
 Product p3 = new Product("P103", "Mouse", 40, 299.00);  
  
 manager.addProduct(p1);  
 manager.addProduct(p2);  
 manager.addProduct(p3);  
 System.*out*.print("\n");  
  
 manager.displayInventory();  
 System.*out*.print("\n");  
  
 manager.updateProduct("P102", 60, 459.00);  
 manager.deleteProduct("P101");  
 System.*out*.print("\n");  
  
 manager.displayInventory();  
 }  
}

**Output :**

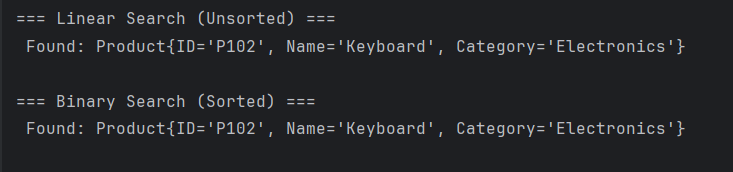
****

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

**Code :**

import java.util.\*;  
  
class Product {  
 String productId;  
 String productName;  
 String category;  
  
 public Product(String id, String name, String category) {  
 this.productId = id;  
 this.productName = name;  
 this.category = category;  
 }  
  
 @Override  
 public String toString() {  
 return "Product{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, 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;  
 }  
  
 public static void main(String[] args) {  
 Product[] products = {  
 new Product("P101", "Mouse", "Electronics"),  
 new Product("P102", "Keyboard", "Electronics"),  
 new Product("P103", "Monitor", "Electronics"),  
 new Product("P104", "Chair", "Furniture"),  
 new Product("P105", "Table", "Furniture")  
 };  
  
 System.*out*.println("=== Linear Search (Unsorted) ===");  
 Product linearResult = *linearSearch*(products, "Keyboard");  
 System.*out*.println(linearResult != null ? " Found: " + linearResult : " Not Found");  
  
  
 Arrays.*sort*(products, Comparator.*comparing*(p -> p.productName));  
  
 System.*out*.println("\n=== Binary Search (Sorted) ===");  
 Product binaryResult = *binarySearch*(products, "Keyboard");  
 System.*out*.println(binaryResult != null ? " Found: " + binaryResult : " Not Found");  
 }  
}

**Output :**

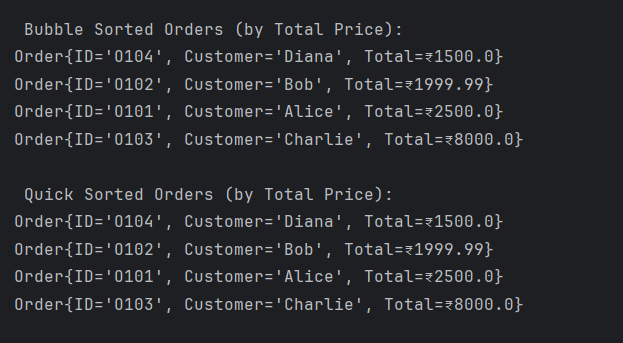
****

**Exercise 3: Sorting Customer Orders**

**Code :**

import java.util.Arrays;  
  
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 "Order{ID='" + orderId + "', Customer='" + customerName + "', Total=₹" + totalPrice + "}";  
 }  
}  
  
public class OrderSorting {  
 public static void bubbleSort(Order[] orders) {  
 int n = orders.length;  
 for (int i = 0; i < n - 1; i++) {  
 boolean swapped = false;  
 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;  
 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);  
 *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(String message, Order[] orders) {  
 System.*out*.println("\n" + message);  
 for (Order o : orders) {  
 System.*out*.println(o);  
 }  
 }  
  
 public static void main(String[] args) {  
 Order[] orders = {  
 new Order("O101", "Alice", 2500.0),  
 new Order("O102", "Bob", 1999.99),  
 new Order("O103", "Charlie", 8000.0),  
 new Order("O104", "Diana", 1500.0)  
 };  
  
 Order[] bubbleSorted = Arrays.*copyOf*(orders, orders.length);  
 Order[] quickSorted = Arrays.*copyOf*(orders, orders.length);  
  
 *bubbleSort*(bubbleSorted);  
 *printOrders*(" Bubble Sorted Orders (by Total Price):", bubbleSorted);  
  
 *quickSort*(quickSorted, 0, quickSorted.length - 1);  
 *printOrders*(" Quick Sorted Orders (by Total Price):", quickSorted);  
 }  
}

**Output :**

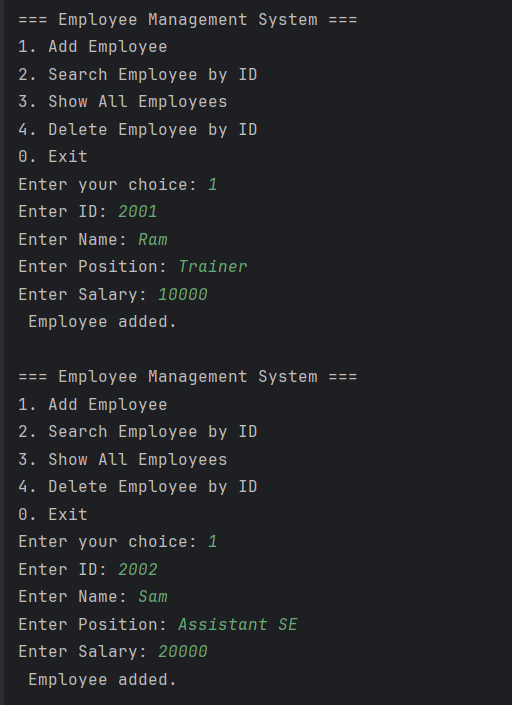
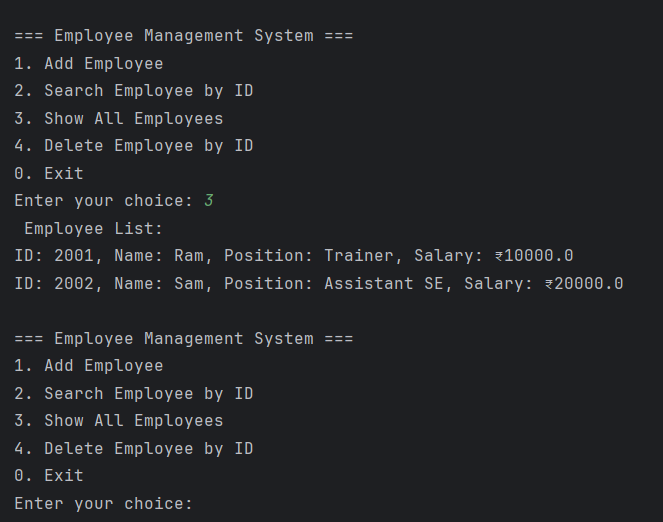
****

**Exercise 4: Employee Management System**

**Code :**

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;  
 this.position = position;  
 this.salary = salary;  
 }  
 public void display() {  
 System.*out*.println("ID: " + employeeId + ", Name: " + name +  
 ", Position: " + position + ", Salary: ₹" + salary);  
 }  
}  
  
public class EmployeeManagementSystem {  
 private static final int *MAX\_EMPLOYEES* = 100;  
 private Employee[] employees = new Employee[*MAX\_EMPLOYEES*];  
 private int count = 0;  
  
 public void addEmployee(Employee emp) {  
 if (count < *MAX\_EMPLOYEES*) {  
 employees[count++] = emp;  
 System.*out*.println(" Employee added.");  
 } else {  
 System.*out*.println(" Cannot add more employees.");  
 }  
 }  
  
 public void searchEmployee(int id) {  
 for (int i = 0; i < count; i++) {  
 if (employees[i].employeeId == id) {  
 System.*out*.println(" Employee Found:");  
 employees[i].display();  
 return;  
 }  
 }  
 System.*out*.println(" Employee ID not found.");  
 }  
  
 public void showAllEmployees() {  
 if (count == 0) {  
 System.*out*.println(" No employees to show.");  
 } else {  
 System.*out*.println(" Employee List:");  
 for (int i = 0; i < count; i++) {  
 employees[i].display();  
 }  
 }  
 }  
  
 public void deleteEmployee(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];  
 }  
 employees[--count] = null;  
 System.*out*.println("️ Employee deleted.");  
 return;  
 }  
 }  
 System.*out*.println(" Employee ID not found.");  
 }  
  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 EmployeeManagementSystem system = new EmployeeManagementSystem();  
 int choice;  
  
 do {  
 System.*out*.println("\n=== Employee Management System ===");  
 System.*out*.println("1. Add Employee");  
 System.*out*.println("2. Search Employee by ID");  
 System.*out*.println("3. Show All Employees");  
 System.*out*.println("4. Delete Employee by ID");  
 System.*out*.println("0. Exit");  
 System.*out*.print("Enter your choice: ");  
 choice = sc.nextInt();  
 sc.nextLine();  
  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter ID: ");  
 int id = sc.nextInt();  
 sc.nextLine();  
 System.*out*.print("Enter Name: ");  
 String name = sc.nextLine();  
 System.*out*.print("Enter Position: ");  
 String position = sc.nextLine();  
 System.*out*.print("Enter Salary: ");  
 double salary = sc.nextDouble();  
 Employee emp = new Employee(id, name, position, salary);  
 system.addEmployee(emp);  
 break;  
  
 case 2:  
 System.*out*.print("Enter ID to search: ");  
 system.searchEmployee(sc.nextInt());  
 break;  
  
 case 3:  
 system.showAllEmployees();  
 break;  
  
 case 4:  
 System.*out*.print("Enter ID to delete: ");  
 system.deleteEmployee(sc.nextInt());  
 break;  
  
 case 0:  
 System.*out*.println(" Exiting...");  
 break;  
  
 default:  
 System.*out*.println(" Invalid choice!");  
 }  
 } while (choice != 0);  
  
 sc.close();  
 }  
}

**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;  
 }

public void display() {  
 System.*out*.println("TaskID: " + taskId + ", Task: " + taskName + ", Status: " + status);  
 }  
}  
  
class TaskNode {  
 Task task;  
 TaskNode next;  
 public TaskNode(Task task) {  
 this.task = task;  
 this.next = null;  
 }  
}  
  
public class TaskManagementSystem {  
 TaskNode head;

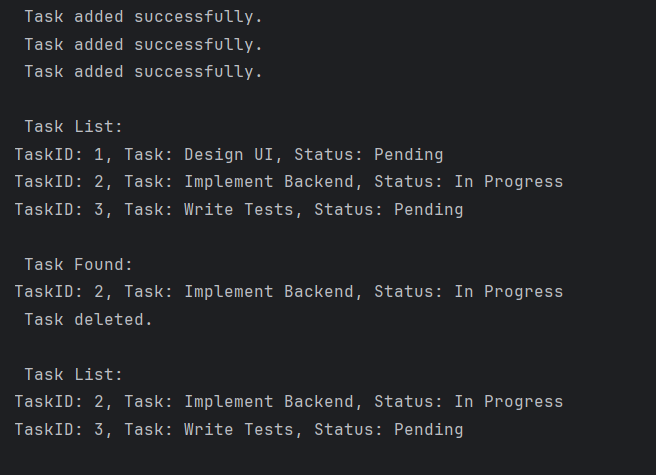
public void addTask(Task task) {  
 TaskNode newNode = new TaskNode(task);  
 if (head == null) {  
 head = newNode;  
 } else {  
 TaskNode temp = head;  
 while (temp.next != null) {  
 temp = temp.next;  
 }  
 temp.next = newNode;  
 }  
 System.*out*.println(" Task added successfully.");  
 }

public void displayTasks() {  
 if (head == null) {  
 System.*out*.println(" No tasks available.");  
 return;  
 }  
 System.*out*.println(" Task List:");  
 TaskNode temp = head;  
 while (temp != null) {  
 temp.task.display();  
 temp = temp.next;  
 }  
 }

public void searchTask(int taskId) {  
 TaskNode temp = head;  
 while (temp != null) {  
 if (temp.task.taskId == taskId) {  
 System.*out*.println(" Task Found:");  
 temp.task.display();  
 return;  
 }  
 temp = temp.next;  
 }  
 System.*out*.println(" Task not found.");  
 }

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;  
 }  
  
 TaskNode temp = head;  
 while (temp.next != null && temp.next.task.taskId != taskId) {  
 temp = temp.next;  
 }  
  
 if (temp.next == null) {  
 System.*out*.println(" Task ID not found.");  
 } else {  
 temp.next = temp.next.next;  
 System.*out*.println(" Task deleted.");  
 }  
 }  
  
 public static void main(String[] args) {  
 TaskManagementSystem system = new TaskManagementSystem();  
  
 system.addTask(new Task(1, "Design UI", "Pending"));  
 system.addTask(new Task(2, "Implement Backend", "In Progress"));  
 system.addTask(new Task(3, "Write Tests", "Pending"));  
 System.*out*.print("\n");  
  
 system.displayTasks();  
 System.*out*.print("\n");  
  
 system.searchTask(2);  
 system.deleteTask(1);  
 System.*out*.print("\n");  
 system.displayTasks();  
 }  
}

**Output :**

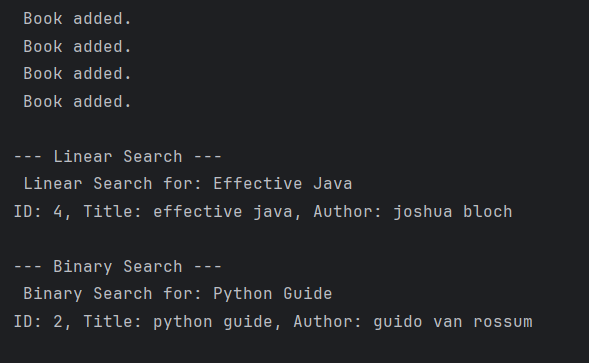
****

**Exercise 6: Library Management System**

**Code :**

import java.util.\*;  
  
class Book implements Comparable<Book> {  
 int bookId;  
 String title;  
 String author;  
  
 public Book(int bookId, String title, String author) {  
 this.bookId = bookId;  
 this.title = title.toLowerCase(); // normalize for comparison  
 this.author = author.toLowerCase();  
 }  
  
 public void display() {  
 System.*out*.println("ID: " + bookId + ", Title: " + title + ", Author: " + author);  
 }  
  
 @Override  
 public int compareTo(Book other) {  
 return this.title.compareTo(other.title);   
 }  
}  
  
public class LibraryManagementSystem {  
  
 List<Book> books = new ArrayList<>();  
  
 public void addBook(Book book) {  
 books.add(book);  
 System.*out*.println(" Book added.");  
 }  
  
 public void linearSearch(String title) {  
 System.*out*.println(" Linear Search for: " + title);  
 boolean found = false;  
 for (Book book : books) {  
 if (book.title.equalsIgnoreCase(title)) {  
 book.display();  
 found = true;  
 }  
 }  
 if (!found) {  
 System.*out*.println(" Book not found.");  
 }  
 }  
  
 public void binarySearch(String title) {  
 System.*out*.println(" Binary Search for: " + title);  
 Collections.*sort*(books);   
  
 int low = 0;  
 int high = books.size() - 1;  
 while (low <= high) {  
 int mid = (low + high) / 2;  
 int compare = books.get(mid).title.compareToIgnoreCase(title);  
 if (compare == 0) {  
 books.get(mid).display();  
 return;  
 } else if (compare < 0) {  
 low = mid + 1;  
 } else {  
 high = mid - 1;  
 }  
 }  
 System.*out*.println(" Book not found.");  
 }  
  
 public static void main(String[] args) {  
 LibraryManagementSystem lib = new LibraryManagementSystem();  
  
 lib.addBook(new Book(1, "Java Basics", "James Gosling"));  
 lib.addBook(new Book(2, "Python Guide", "Guido van Rossum"));  
 lib.addBook(new Book(3, "C++ Fundamentals", "Bjarne Stroustrup"));  
 lib.addBook(new Book(4, "Effective Java", "Joshua Bloch"));  
  
 System.*out*.println("\n--- Linear Search ---");  
 lib.linearSearch("Effective Java");  
  
 System.*out*.println("\n--- Binary Search ---");  
 lib.binarySearch("Python Guide");  
 }  
}

**Output :**



**Exercise 7: Financial Forecasting**

**Code :**

public class FinancialForecasting {  
 public static double forecastRecursive(double presentValue, double rate, int years) {  
 if (years == 0)  
 return presentValue;  
 return *forecastRecursive*(presentValue, rate, years - 1) \* (1 + rate);  
 }  
  
 public static double forecastMemoized(double presentValue, double rate, int years, double[] memo) {  
 if (years == 0) return presentValue;  
 if (memo[years] != 0) return memo[years];  
 return memo[years] = *forecastMemoized*(presentValue, rate, years - 1, memo) \* (1 + rate);  
 }  
  
 public static double forecastIterative(double presentValue, double rate, int years) {  
 double result = presentValue;  
 for (int i = 1; i <= years; i++) {  
 result \*= (1 + rate);  
 }  
 return result;  
 }  
  
 public static void printForecastTable(double presentValue, double rate, int totalYears) {  
 System.*out*.println("\n Forecast Table (Iterative):");  
 System.*out*.printf("%-5s %-15s\n", "Year", "Future Value (₹)");  
 double value = presentValue;  
 for (int i = 0; i <= totalYears; i++) {  
 System.*out*.printf("%-5d ₹%-15.2f\n", i, value);  
 value \*= (1 + rate);  
 }  
 }  
  
 public static void main(String[] args) {  
 double presentValue = 10000;  
 double growthRate = 0.08;  
 int years = 5;  
  
 System.*out*.println(" Recursive Result:");  
 System.*out*.printf("Year %d: ₹%.2f\n", years, *forecastRecursive*(presentValue, growthRate, years));  
  
 System.*out*.println("\n Memoized Result:");  
 double[] memo = new double[years + 1];  
 System.*out*.printf("Year %d: ₹%.2f\n", years, *forecastMemoized*(presentValue, growthRate, years, memo));  
  
 System.*out*.println("\n Iterative Result:");  
 System.*out*.printf("Year %d: ₹%.2f\n", years, *forecastIterative*(presentValue, growthRate, years));  
  
 *printForecastTable*(presentValue, growthRate, years);  
 }  
}

**Output :**

