**ALGORITHMS AND DATA STRUCTURES**

1. **Inventory Management System**

**Code:**

**// Product.java**

public 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;

}

// Getters and Setters

public String getProductId() { return productId; }

public String getProductName() { return productName; }

public int getQuantity() { return quantity; }

public double getPrice() { return price; }

public void setQuantity(int quantity) { this.quantity = quantity; }

public void setPrice(double price) { this.price = price; }

@Override

public String toString() {

return "ID: " + productId + ", Name: " + productName + ", Qty: " + quantity + ", Price: Rs." + price;

}

}

// InventoryManager.java

import java.util.HashMap;

import java.util.Map;

public class InventoryManager {

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

// Add new product

public void addProduct(Product product) {

if (inventory.containsKey(product.getProductId())) {

System.out.println("Product ID already exists.");

return;

}

inventory.put(product.getProductId(), product);

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

}

// Update existing product

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

Product p = inventory.get(productId);

if (p != null) {

p.setQuantity(quantity);

p.setPrice(price);

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

} else {

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

}

}

// Delete product

public void deleteProduct(String productId) {

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

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

} else {

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

}

}

// Display all products

public void displayInventory() {

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

System.out.println(p);

}

}

}

// InventoryTest.java

public class InventoryTest {

public static void main(String[] args) {

InventoryManager manager = new InventoryManager();

Product p1 = new Product("P101", "Laptop", 10, 55000);

Product p2 = new Product("P102", "Mouse", 50, 499);

Product p3 = new Product("P103", "Keyboard", 30, 999);

manager.addProduct(p1);

manager.addProduct(p2);

manager.addProduct(p3);

manager.displayInventory();

manager.updateProduct("P102", 60, 475);

manager.deleteProduct("P103");

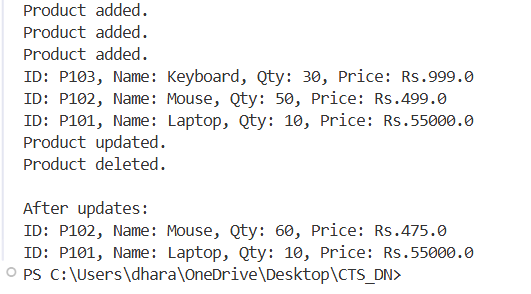
System.out.println("\nAfter updates:");

manager.displayInventory();

}

}

**Output:**



1. **E-commerce Platform Search Function**

**Code:**

**// Product.java**

public class Product {

private String productId;

private String productName;

private String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String getProductId() { return productId; }

public String getProductName() { return productName; }

public String getCategory() { return category; }

@Override

public String toString() {

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

}

}

**// SearchUtils.java**

public class SearchUtils {

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

for (Product product : products) {

if (product.getProductName().equalsIgnoreCase(name)) {

return product;

}

}

return null;

}

// Binary Search

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

int low = 0;

int high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

String midName = products[mid].getProductName();

int cmp = name.compareToIgnoreCase(midName);

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

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

else low = mid + 1;

}

return null;

}

}

**// ECommerceSearchTest.java**

import java.util.Arrays;

import java.util.Comparator;

public class ECommerceSearchTest {

public static void main(String[] args) {

Product[] products = {

new Product("P101", "Mouse", "Electronics"),

new Product("P102", "Laptop", "Electronics"),

new Product("P103", "Notebook", "Stationery"),

new Product("P104", "Pen", "Stationery"),

new Product("P105", "Headphones", "Electronics")

};

// Linear Search

Product result1 = SearchUtils.linearSearch(products, "Pen");

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

// Sort array for binary search (by productName)

Arrays.sort(products, Comparator.comparing(Product::getProductName));

// Binary Search

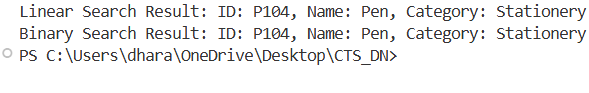
Product result2 = SearchUtils.binarySearch(products, "Pen");

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

}

}

Output:



1. **Sorting Customer Orders**

**Code:**

**// Order.java**

public class Order {

private String orderId;

private String customerName;

private double totalPrice;

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

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public double getTotalPrice() {

return totalPrice;

}

@Override

public String toString() {

return "OrderID: " + orderId + ", Customer: " + customerName + ", Total: Rs." + totalPrice;

}

}

**// SortUtils.java**

public class SortUtils {

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].getTotalPrice() > orders[j + 1].getTotalPrice()) {

Order temp = orders[j];

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

orders[j + 1] = temp;

swapped = true;

}

}

if (!swapped) break; // Optimization

}

}

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].getTotalPrice();

int i = low - 1;

for (int j = low; j < high; j++) {

if (orders[j].getTotalPrice() <= 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;

}

}

**// SortTest.java**

public class SortTest {

public static void main(String[] args) {

Order[] orders1 = {

new Order("O1", "Alice", 1200.0),

new Order("O2", "Bob", 540.0),

new Order("O3", "Charlie", 2750.5),

new Order("O4", "Daisy", 1999.9),

new Order("O5", "Eve", 899.0)

};

// Copy for quick sort

Order[] orders2 = orders1.clone();

// Bubble Sort

System.out.println("== Bubble Sort ==");

SortUtils.bubbleSort(orders1);

for (Order o : orders1) System.out.println(o);

// Quick Sort

System.out.println("\n== Quick Sort ==");

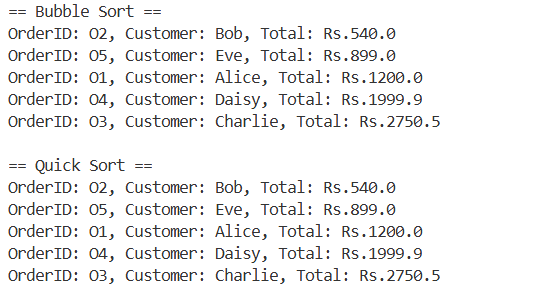
SortUtils.quickSort(orders2, 0, orders2.length - 1);

for (Order o : orders2) System.out.println(o);

}

}

**Output:**



1. **Employee Management System**

**Code:**

**// Employee.java**

public class Employee {

private String employeeId;

private String name;

private String position;

private double salary;

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

this.employeeId = employeeId;

this.name = name;

this.position = position;

this.salary = salary;

}

public String getEmployeeId() {

return employeeId;

}

@Override

public String toString() {

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

}

}

**// EmployeeManager.java**

public class EmployeeManager {

private Employee[] employees;

private int size;

public EmployeeManager(int capacity) {

employees = new Employee[capacity];

size = 0;

}

// Add employee

public void addEmployee(Employee emp) {

if (size >= employees.length) {

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

return;

}

employees[size++] = emp;

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

}

// Search employee by ID

public Employee searchEmployee(String empId) {

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

if (employees[i].getEmployeeId().equals(empId)) {

return employees[i];

}

}

return null;

}

// Traverse all employees

public void displayAllEmployees() {

if (size == 0) {

System.out.println("No employees found.");

return;

}

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

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

}

}

// Delete employee by ID

public void deleteEmployee(String empId) {

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

if (employees[i].getEmployeeId().equals(empId)) {

// Shift elements left to fill the gap

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

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

}

employees[--size] = null;

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

return;

}

}

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

}

}

**// EmployeeSystemTest.java**

public class EmployeeSystemTest {

public static void main(String[] args) {

EmployeeManager manager = new EmployeeManager(5);

// Add employees

manager.addEmployee(new Employee("E101", "Alice", "Manager", 75000));

manager.addEmployee(new Employee("E102", "Bob", "Engineer", 55000));

manager.addEmployee(new Employee("E103", "Charlie", "Designer", 50000));

// Display all

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

manager.displayAllEmployees();

// Search employee

System.out.println("\nSearch for E102:");

Employee found = manager.searchEmployee("E102");

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

// Delete employee

System.out.println("\nDeleting E102:");

manager.deleteEmployee("E102");

// Display all again

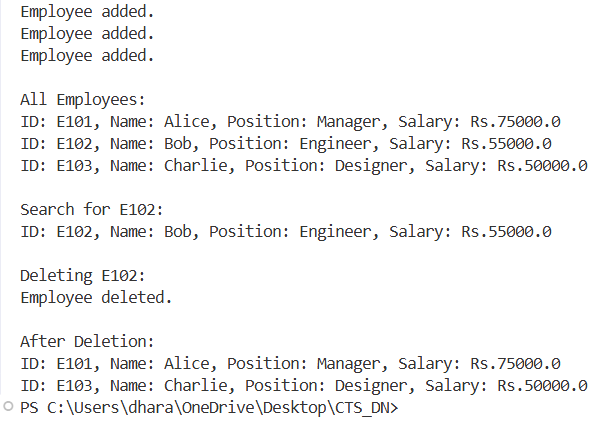
System.out.println("\nAfter Deletion:");

manager.displayAllEmployees();

}

}

**Output:**



1. **Task Management System**

**Code:**

**// Task.java**

public class Task {

private String taskId;

private String taskName;

private String status;

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

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

}

public String getTaskId() {

return taskId;

}

@Override

public String toString() {

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

}

}

**// Node.java**

public class Node {

Task task;

Node next;

public Node(Task task) {

this.task = task;

this.next = null;

}

}

**// TaskManager.java**

public class TaskManager {

private Node head;

// Add task to end

public void addTask(Task task) {

Node newNode = new Node(task);

if (head == null) {

head = newNode;

} else {

Node curr = head;

while (curr.next != null) {

curr = curr.next;

}

curr.next = newNode;

}

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

}

// Search by ID

public Task searchTask(String taskId) {

Node curr = head;

while (curr != null) {

if (curr.task.getTaskId().equals(taskId)) {

return curr.task;

}

curr = curr.next;

}

return null;

}

// Delete by ID

public void deleteTask(String taskId) {

if (head == null) {

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

return;

}

if (head.task.getTaskId().equals(taskId)) {

head = head.next;

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

return;

}

Node curr = head;

while (curr.next != null && !curr.next.task.getTaskId().equals(taskId)) {

curr = curr.next;

}

if (curr.next != null) {

curr.next = curr.next.next;

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

} else {

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

}

}

// Display all tasks

public void displayTasks() {

Node curr = head;

if (curr == null) {

System.out.println("No tasks found.");

return;

}

while (curr != null) {

System.out.println(curr.task);

curr = curr.next;

}

}

}

**// TaskSystemTest.java**

public class TaskSystemTest {

public static void main(String[] args) {

TaskManager manager = new TaskManager();

// Adding tasks

manager.addTask(new Task("T1", "Design Database", "Pending"));

manager.addTask(new Task("T2", "Implement API", "In Progress"));

manager.addTask(new Task("T3", "Write Tests", "Pending"));

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

manager.displayTasks();

// Searching for a task

System.out.println("\nSearch Task T2:");

Task result = manager.searchTask("T2");

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

// Deleting a task

System.out.println("\nDeleting Task T2:");

manager.deleteTask("T2");

// Display after deletion

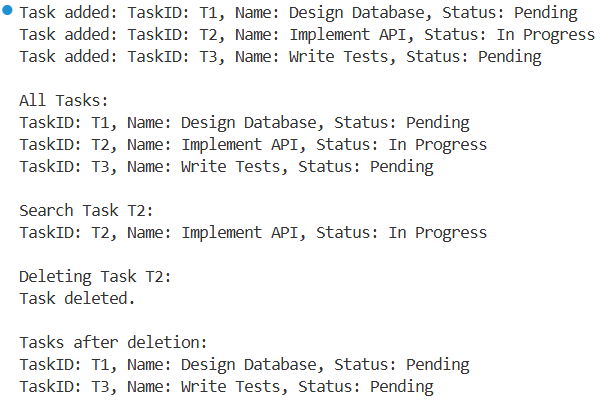
System.out.println("\nTasks after deletion:");

manager.displayTasks();

}

}

**Output:**



1. **Library Management System**

**Code:**

**// Book.java**

public class Book {

private String bookId;

private String title;

private String author;

public Book(String bookId, String title, String author) {

this.bookId = bookId;

this.title = title;

this.author = author;

}

public String getTitle() {

return title;

}

@Override

public String toString() {

return "BookID: " + bookId + ", Title: " + title + ", Author: " + author;

}

}

**// SearchUtil.java**

import java.util.Arrays;

import java.util.Comparator;

public class SearchUtil {

// Linear search by title

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

for (Book b : books) {

if (b.getTitle().equalsIgnoreCase(title)) {

return b;

}

}

return null;

}

// Binary search by title (assuming sorted)

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

int low = 0, high = books.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

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

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

else low = mid + 1;

}

return null;

}

// Sort by title before binary search

public static void sortByTitle(Book[] books) {

Arrays.sort(books, Comparator.comparing(Book::getTitle, String.CASE\_INSENSITIVE\_ORDER));

}

}

**// LibrarySearchTest.java**

public class LibrarySearchTest {

public static void main(String[] args) {

Book[] books = {

new Book("B101", "Data Structures", "Mark Allen"),

new Book("B102", "Operating Systems", "Silberschatz"),

new Book("B103", "Computer Networks", "Tanenbaum"),

new Book("B104", "Java Programming", "Herbert Schildt"),

new Book("B105", "Database Systems", "Raghu Ramakrishnan")

};

// Linear search

System.out.println("== Linear Search ==");

Book result1 = SearchUtil.linearSearch(books, "Java Programming");

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

// Binary search (after sorting)

SearchUtil.sortByTitle(books);

System.out.println("\n== Binary Search ==");

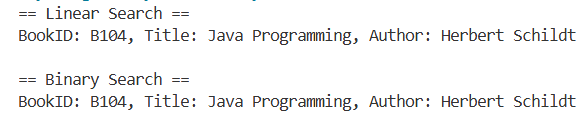
Book result2 = SearchUtil.binarySearch(books, "Java Programming");

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

}

}

**Output:**



1. **Financial Forecasting**

**Code:**

**// ForecastCalculator.java**

public class ForecastCalculator {

// Recursive method to compute future value

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

if (years == 0) return initialValue;

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

}

}

**// ForecastTest.java**

public class ForecastTest {

public static void main(String[] args) {

double initialValue = 10000; // ₹10,000

double rate = 0.10; // 10% growth per year

int years = 5;

double result = ForecastCalculator.predictFutureValue(initialValue, rate, years);

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

}

}

**Output:**

