**Module 2 - Data Structures and Algorithms**

**Exercise 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;

}

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;

}

}

Inventory.java

import java.util.HashMap;

public class Inventory {

private HashMap<String, Product> products;

public Inventory() {

products = new HashMap<>();

}

public void addProduct(Product product) {

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

}

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

Product product = products.get(productId);

if (product != null) {

product.setQuantity(quantity);

product.setPrice(price);

}

}

public void deleteProduct(String productId) {

products.remove(productId);

}

// Method to retrieve a product

public Product getProduct(String productId) {

return products.get(productId);

}

}

InventoryManagementTest.java

public class InventoryManagementTest {

public static void main(String[] args) {

Inventory inventory = new Inventory();

Product product1 = new Product("P001", "Laptop", 10, 999.99);

Product product2 = new Product("P002", "Smartphone", 20, 499.99);

Product product3 = new Product("P003", "Tablet", 15, 299.99);

inventory.addProduct(product1);

inventory.addProduct(product2);

inventory.addProduct(product3);

System.out.println("Products added to inventory:");

System.out.println(inventory.getProduct("P001").getProductName());

System.out.println(inventory.getProduct("P002").getProductName());

System.out.println(inventory.getProduct("P003").getProductName());

System.out.println("\nUpdating product P002...");

inventory.updateProduct("P002", 18, 479.99); // Update quantity and price

Product updatedProduct = inventory.getProduct("P002");

System.out.println("Updated Product: " + updatedProduct.getProductName() + ", Quantity: " + updatedProduct.getQuantity() + ", Price: $" + updatedProduct.getPrice());

System.out.println("\nDeleting product P003...");

inventory.deleteProduct("P003");

if (inventory.getProduct("P003") == null) {

System.out.println("Product P003 has been successfully deleted.");

} else {

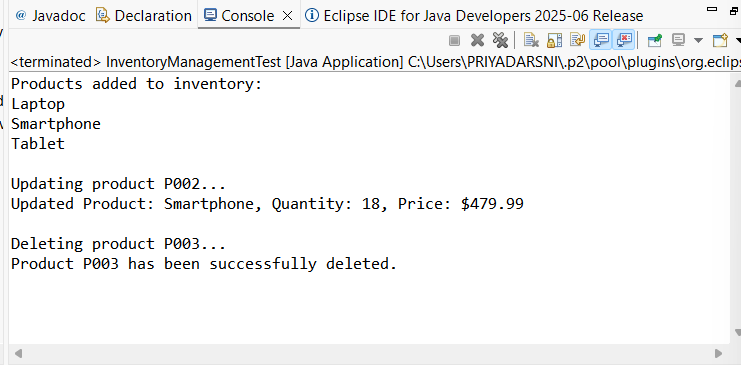
System.out.println("Product P003 still exists in the inventory.");

}

}

}

OUTPUT:



**Exercise 2: 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;

}

}

SearchUtil.java

public class SearchUtil {

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

for (Product product : products) {

if (product.getProductId().equals(productId)) {

return product;

}

}

return null;

}

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

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

if (products[mid].getProductId().equals(productId)) {

return products[mid]; // Product found

}

if (products[mid].getProductId().compareTo(productId) < 0) {

left = mid + 1; // Search in the right half

} else {

right = mid - 1;

}

}

return null;

}

}

Main.java

public class Main {

public static void main(String[] args) {

Product[] products = {

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

new Product("P002", "Smartphone", "Electronics"),

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

new Product("P004", "Headphones", "Accessories"),

new Product("P005", "Smartwatch", "Wearables")

};

Product foundProductLinear = SearchUtil.linearSearch(products, "P003");

if (foundProductLinear != null) {

System.out.println("Linear Search: Found " + foundProductLinear.getProductName());

} else {

System.out.println("Linear Search: Product not found.");

}

Product[] sortedProducts = {

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

new Product("P002", "Smartphone", "Electronics"),

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

new Product("P004", "Headphones", "Accessories"),

new Product("P005", "Smartwatch", "Wearables")

};

Product foundProductBinary = SearchUtil.binarySearch(sortedProducts, "P003");

if (foundProductBinary != null) {

System.out.println("Binary Search: Found " + foundProductBinary.getProductName());

} else {

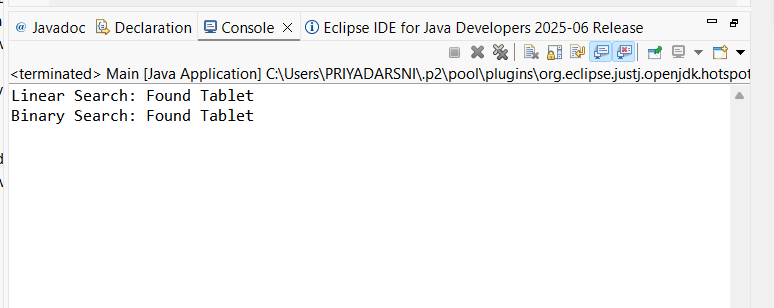
System.out.println("Binary Search: Product not found.");

}

}

}

OUTPUT:



**Exercise 3: 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 String getOrderId() {

return orderId;

}

public String getCustomerName() {

return customerName;

}

public double getTotalPrice() {

return totalPrice;

}

}

SortUtil.java

public class SortUtil {

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

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].getTotalPrice();

int i = (low - 1); // Index of smaller element

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

if (orders[j].getTotalPrice() <= pivot) {

i++;

// Swap orders[i] and orders[j]

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;

}

}

OrderSortingTest.java

public class OrderSortingTest {

public static void main(String[] args) {

Order[] orders = {

new Order("O001", "Alice", 250.75),

new Order("O002", "Bob", 150.50),

new Order("O003", "Charlie", 300.00),

new Order("O004", "David", 100.25),

new Order("O005", "Eve", 200.00)

};

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

displayOrders(orders);

System.out.println("\nSorting using Bubble Sort...");

SortUtil.bubbleSort(orders);

System.out.println("Orders sorted by total price (Bubble Sort):");

displayOrders(orders);

Order[] ordersForQuickSort = {

new Order("O001", "Alice", 250.75),

new Order("O002", "Bob", 150.50),

new Order("O003", "Charlie", 300.00),

new Order("O004", "David", 100.25),

new Order("O005", "Eve", 200.00)

};

System.out.println("\nSorting using Quick Sort...");

SortUtil.quickSort(ordersForQuickSort, 0, ordersForQuickSort.length - 1);

System.out.println("Orders sorted by total price (Quick Sort):");

displayOrders(ordersForQuickSort);

}

private static void displayOrders(Order[] orders) {

for (Order order : orders) {

System.out.printf("Order ID: %s, Customer: %s, Total Price: $%.2f%n",

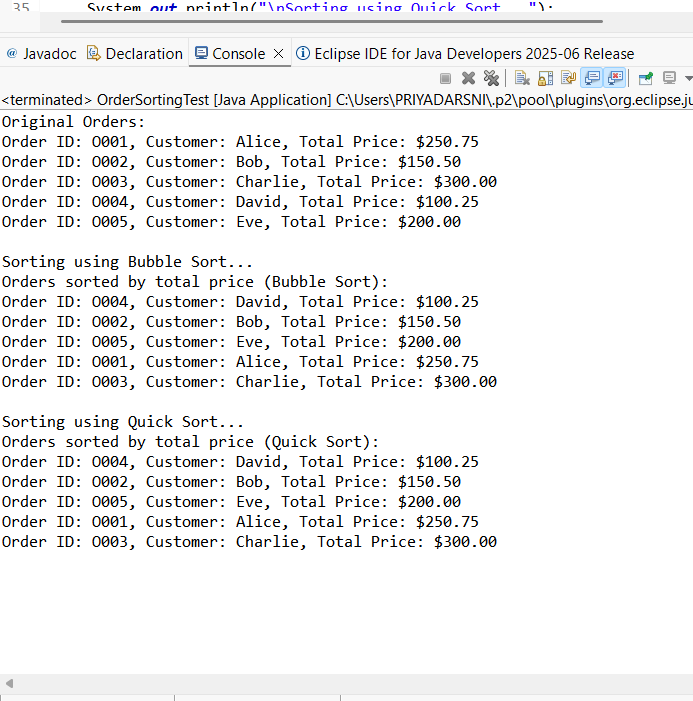
order.getOrderId(), order.getCustomerName(), order.getTotalPrice());

}

}

}

OUTPUT:



**Exercise 4: Employee Management System**

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;

}

public String getName() {

return name;

}

public String getPosition() {

return position;

}

public double getSalary() {

return salary;

}

@Override

public String toString() {

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

}

}

EmployeeManagement.java

public class EmployeeManagement {

private Employee[] employees;

private int count;

public EmployeeManagement(int size) {

employees = new Employee[size];

count = 0;

} public void addEmployee(Employee employee) {

if (count < employees.length) {

employees[count] = employee;

count++;

} else {

System.out.println("Employee array is full. Cannot add more employees.");

}

}

public Employee searchEmployee(String employeeId) {

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

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

return employees[i];

}

}

return null; }

public void traverseEmployees() {

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

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

}

}

public void deleteEmployee(String employeeId) {

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

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

// Shift elements to the left

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

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

}

employees[count - 1] = null; // Clear the last element

count--;

System.out.println("Employee with ID " + employeeId + " has been deleted.");

return;

}

}

System.out.println("Employee with ID " + employeeId + " not found.");

}

}

EmployeeManagementTest.java

public class EmployeeManagementTest {

public static void main(String[] args) {

// Create an instance of EmployeeManagement with a capacity of 5

EmployeeManagement employeeManagement = new EmployeeManagement(5);

Employee emp1 = new Employee("E001", "Alice", "Developer", 75000);

Employee emp2 = new Employee("E002", "Bob", "Designer", 65000);

Employee emp3 = new Employee("E003", "Charlie", "Manager", 85000);

Employee emp4 = new Employee("E004", "David", "Analyst", 60000);

Employee emp5 = new Employee("E005", "Eve", "Tester", 55000);

employeeManagement.addEmployee(emp1);

employeeManagement.addEmployee(emp2);

employeeManagement.addEmployee(emp3);

employeeManagement.addEmployee(emp4);

employeeManagement.addEmployee(emp5);

Employee emp6 = new Employee("E006", "Frank", "Support", 50000);

employeeManagement.addEmployee(emp6); // This should print an error message

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

employeeManagement.traverseEmployees();

System.out.println("\nSearching for employee with ID E003...");

Employee foundEmployee = employeeManagement.searchEmployee("E003");

if (foundEmployee != null) {

System.out.println("Found: " + foundEmployee);

} else {

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

}

System.out.println("\nDeleting employee with ID E002...");

employeeManagement.deleteEmployee("E002");

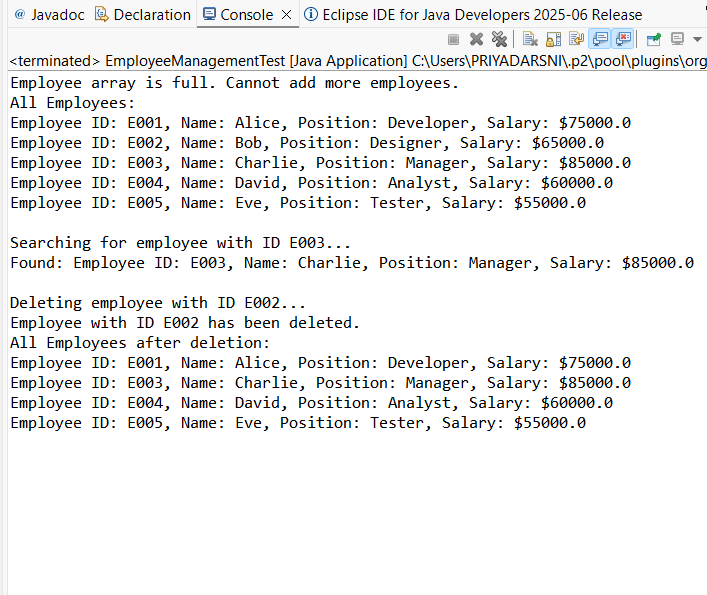
System.out.println("All Employees after deletion:");

employeeManagement.traverseEmployees();

}

}

OUTPUT:



**Exercise 5: 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;

}

public String getTaskName() {

return taskName;

}

public String getStatus() {

return status;

}

public String toString() {

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

}

}

TaskNode.java

class TaskNode {

Task task;

TaskNode next;

public TaskNode(Task task) {

this.task = task;

this.next = null;

}

}

TaskManagement.java

public class TaskManagement {

private TaskNode head;

public void addTask(Task task) {

TaskNode newNode = new TaskNode(task);

if (head == null) {

head = newNode; // If the list is empty, set head to new node

} else {

TaskNode current = head;

while (current.next != null) {

current = current.next; }

current.next = newNode

}

}

public Task searchTask(String taskId) {

TaskNode current = head;

while (current != null) {

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

return current.task;

}

current = current.next;

}

return null;

}

public void traverseTasks() {

TaskNode current = head;

while (current != null) {

System.out.println(current.task);

current = current.next; }

}

public void deleteTask(String taskId) {

if (head == null) {

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

return;

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

head = head.next; // Move head to the next node

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

return;

}

TaskNode current = head;

while (current.next != null) {

if (current.next.task.getTaskId().equals(taskId)) {

current.next = current.next.next; // Bypass the node to delete it

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

return;

}

current = current.next; // Move to the next node

}

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

}

}

TaskManagementTest.java

public class TaskManagementTest {

public static void main(String[] args) {

TaskManagement taskManagement = new TaskManagement();

Task task1 = new Task("T001", "Design Homepage", "Pending");

Task task2 = new Task("T002", "Develop Backend", "In Progress");

Task task3 = new Task("T003", "Test Application", "Pending");

Task task4 = new Task("T004", "Deploy to Production", "Pending");

taskManagement.addTask(task1);

taskManagement.addTask(task2);

taskManagement.addTask(task3);

taskManagement.addTask(task4);

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

taskManagement.traverseTasks();

System.out.println("\nSearching for task with ID T002...");

Task foundTask = taskManagement.searchTask("T002");

if (foundTask != null) {

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

} else {

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

}

System.out.println("\nDeleting task with ID T003...");

taskManagement.deleteTask("T003");

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

taskManagement.traverseTasks();

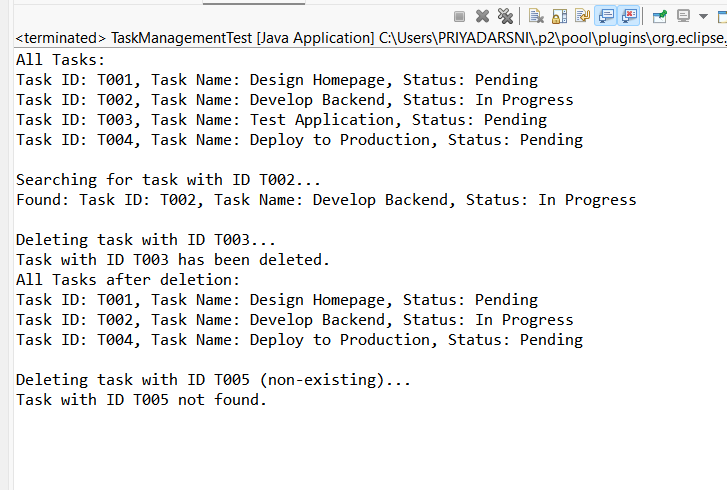
System.out.println("\nDeleting task with ID T005 (non-existing)...");

taskManagement.deleteTask("T005");

}

}

OUTPUT:



**Exercise 6: 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 getBookId() {

return bookId;

}

public String getTitle() {

return title;

}

public String getAuthor() {

return author;

}

public String toString() {

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

}

}

LibraryManagement.java

import java.util.List;

public class LibraryManagement {

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

for (Book book : books) {

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

return book;

}

}

return null;

}

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

int low = 0;

int high = books.size() - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

int comparison = books.get(mid).getTitle().compareToIgnoreCase(title);

if (comparison == 0) {

return books.get(mid);

} else if (comparison < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

}

LibraryManagementTest.java

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class LibraryManagementTest {

public static void main(String[] args) {

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

books.add(new Book("B001", "The Great Gatsby", "F. Scott Fitzgerald"));

books.add(new Book("B002", "To Kill a Mockingbird", "Harper Lee"));

books.add(new Book("B003", "1984", "George Orwell"));

books.add(new Book("B004", "Pride and Prejudice", "Jane Austen"));

books.add(new Book("B005", "The Catcher in the Rye", "J.D. Salinger"));

String searchTitleLinear = "1984";

System.out.println("Linear Search for: " + searchTitleLinear);

Book foundBookLinear = LibraryManagement.linearSearch(books, searchTitleLinear);

if (foundBookLinear != null) {

System.out.println("Found: " + foundBookLinear);

} else {

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

}

Collections.sort(books, (b1, b2) -> b1.getTitle().compareToIgnoreCase(b2.getTitle()));

String searchTitleBinary = "Pride and Prejudice";

System.out.println("\nBinary Search for: " + searchTitleBinary);

Book foundBookBinary = LibraryManagement.binarySearch(books, searchTitleBinary);

if (foundBookBinary != null) {

System.out.println("Found: " + foundBookBinary);

} else {

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

}

String searchTitleNotFound = "Moby Dick";

System.out.println("\nSearching for a non-existing book: " + searchTitleNotFound);

Book notFoundBook = LibraryManagement.linearSearch(books, searchTitleNotFound);

if (notFoundBook != null) {

System.out.println("Found: " + notFoundBook);

} else {

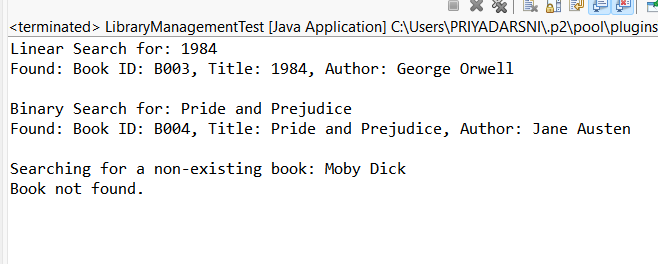
System.out.println("Book not found.");

}

}

}

OUTPUT:



**Exercise 7: Financial Forecasting**

FinancialForecasting.java

public class FinancialForecasting {

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

// Base case: if no years left, return the present value

if (years == 0) {

return presentValue;

}

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

}

public static void main(String[] args) {

double presentValue = 1000.0;

double growthRate = 0.05;

int years = 10;

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

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

}

}

FinancialForecastingWithMemoization.java

public class FinancialForecastingWithMemoization {

private static HashMap<Integer, Double> memo = new HashMap<>();

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

if (memo.containsKey(years)) {

return memo.get(years);

}

if (years == 0) {

return presentValue;

}

double futureValue = calculateFutureValue(presentValue \* (1 + growthRate), growthRate, years - 1);

memo.put(years, futureValue);

return futureValue;

}

public static void main(String[] args) {

double presentValue = 1000.0; // Initial investment

double growthRate = 0.05; // 5% growth rate

int years = 10; // Number of years to forecast

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

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

}

}

OUTPUT:

