Exercise 1: Inventory Management System

```
public class Product {

  private String productId;

  private String productName;

  private int quantity;

  private double price;



  // Constructor

  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 void setProductId(String productId) {

    this.productId = productId;

  }



  public String getProductName() {
    return productName;
}

public void setProductName(String productName) {
    this.productName = productName;
}

public int getQuantity() {
    return quantity;
}

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

public double getPrice() {
    return price;
}

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

@Override
public String toString() {
    return "Product{" +
                "productId='" + productId + '\'' +
                ", productName='" + productName + '\'' +
                ", quantity=" + quantity +
                ", price=" + price +
                '}';
    }
}

import java.util.HashMap;
import java.util.Map;

public class InventoryManager {
    private Map<String, Product> inventory = new HashMap<>();

    // Add a product
    public void addProduct(Product product) {
        inventory.put(product.getProductId(), product);
    }

    // Update a product
    public void updateProduct(String productId, Product updatedProduct) {
        if (inventory.containsKey(productId)) {
            inventory.put(productId, updatedProduct);
        } else {
            System.out.println("Product with ID " + productId + " not found.");
        }
    }

    // Delete a product
    public void deleteProduct(String productId) {
        if (inventory.containsKey(productId)) {
            inventory.remove(productId);
        } else {
            System.out.println("Product with ID " + productId + " not found.");
        }
    }

    // Retrieve a product
    public Product getProduct(String productId) {
        return inventory.get(productId);
    }

    // Print all products
    public void printAllProducts() {
        for (Product product : inventory.values()) {
            System.out.println(product);
        }
    }
}
```

Exercise 2: E-commerce Platform Search Function

Create a Product class with attributes that can be used for searching:

java

```
public class Product {
    private String productId;
    private String productName;
    private String category;

    // Constructor
    public Product(String productId, String productName, String category) {
        this.productId = productId;
        this.productName = productName;
        this.category = category;
    }

    // Getters
    public String getProductId() {
        return productId;
    }

    public String getProductName() {
        return productName;
    }

    public String getCategory() {
        return category;
    }

    @Override
    public String toString() {
        return "Product{" +
                "productId='" + productId + '\'' +
                ", productName='" + productName + '\'' +
                ", category='" + category + '\'' +
                '}';
    }
}
```

3. Implementation

Linear Search Implementation:

```
public class SearchUtil {

    // Linear Search
    public static Product linearSearch(Product[] products, String searchTerm) {
        for (Product product : products) {
            if (product.getProductId().equals(searchTerm) ||
                product.getProductName().equals(searchTerm) ||
                product.getCategory().equals(searchTerm)) {
                return product;
            }
        }
        return null; // Not found
    }
}
```

Binary Search Implementation:

For binary search, we need the array to be sorted. Here, we'll sort by productId.

```
import java.util.Arrays;
import java.util.Comparator;

public class SearchUtil {

    // Binary Search
    public static Product binarySearch(Product[] products, String searchTerm) {
        int low = 0;
        int high = products.length - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;
            Product midProduct = products[mid];

            if (midProduct.getProductId().equals(searchTerm)) {
                return midProduct;
            } else if (midProduct.getProductId().compareTo(searchTerm) < 0) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return null; // Not found
    }

    // Helper method to sort products by productId
    public static void sortProductsById(Product[] products) {
        Arrays.sort(products, Comparator.comparing(Product::getProductId));
    }
}
```

Complete Code Example

java

```
import java.util.Arrays;
import java.util.Comparator;

public class Main {
    public static void main(String[] args) {
        // Create some products
        Product[] products = new Product[]{
            new Product("P001", "Laptop", "Electronics"),
            new Product("P002", "Smartphone", "Electronics"),
            new Product("P003", "Desk Chair", "Furniture"),
            new Product("P004", "Headphones", "Electronics")
        };

        // Linear Search Example
        Product resultLinear = SearchUtil.linearSearch(products, "Smartphone");
        System.out.println("Linear Search Result: " + resultLinear);

        // Sort products by productId for binary search
        SearchUtil.sortProductsById(products);

        // Binary Search Example
        Product resultBinary = SearchUtil.binarySearch(products, "P003");
        System.out.println("Binary Search Result: " + resultBinary);
    }
}
```

Exercise 3: Sorting Customer Orders

Create an Order class with attributes orderId, customerName, and totalPrice.

```
public class Order {
    private String orderId;
    private String customerName;
    private double totalPrice;

    // Constructor
    public Order(String orderId, String customerName, double totalPrice) {
        this.orderId = orderId;
        this.customerName = customerName;
        this.totalPrice = totalPrice;
    }

    // Getters
    public String getOrderId() {
        return orderId;
    }

    public String getCustomerName() {
        return customerName;
    }

    public double getTotalPrice() {
        return totalPrice;
    }

    @Override
    public String toString() {
        return "Order{" +
                "orderId='" + orderId + '\'' +
                ", customerName='" + customerName + '\'' +
                ", totalPrice=" + totalPrice +
                '}';
    }
}
```

3. Implementation

Bubble Sort Implementation:

java

Copy code

```
public class SortingUtil {

    // Bubble Sort
    public static void bubbleSort(Order[] orders) {
        int n = orders.length;
        boolean swapped;
        for (int i = 0; i < n - 1; i++) {
            swapped = false;
            for (int j = 0; j < n - i - 1; j++) {
                if (orders[j].getTotalPrice() > orders[j + 1].getTotalPrice()) {
                    // Swap orders[j] and orders[j + 1]
                    Order temp = orders[j];
                    orders[j] = orders[j + 1];
                    orders[j + 1] = temp;
                    swapped = true;
                }
            }
            // If no two elements were swapped by inner loop, then break
            if (!swapped) break;
        }
    }
}
```

Quick Sort Implementation:

java

```
public class SortingUtil {

    // 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].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;
            }
        }
        // Swap orders[i + 1] and orders[high] (or pivot)
        Order temp = orders[i + 1];
        orders[i + 1] = orders[high];
        orders[high] = temp;

        return i + 1;
    }
}
```

Complete Code Example

```
public class Main {
    public static void main(String[] args) {
        // Create some orders
        Order[] orders = new Order[]{
            new Order("O001", "Alice", 250.75),
            new Order("O002", "Bob", 150.50),
            new Order("O003", "Charlie", 300.00),
            new Order("O004", "David", 100.00)
        };

        // Bubble Sort Example
        Order[] bubbleSortedOrders = orders.clone();
        SortingUtil.bubbleSort(bubbleSortedOrders);
        System.out.println("Bubble Sorted Orders:");
        for (Order order : bubbleSortedOrders) {
            System.out.println(order);
        }

        // Quick Sort Example
        Order[] quickSortedOrders = orders.clone();
        SortingUtil.quickSort(quickSortedOrders, 0, quickSortedOrders.length - 1);
        System.out.println("Quick Sorted Orders:");
        for (Order order : quickSortedOrders) {
            System.out.println(order);
        }
    }
}
```

Exercise 4: Employee Management System

Create an Employee class with attributes like employeeId, name, position, and salary.

java

```
public class Employee {
    private String employeeId;
    private String name;
    private String position;
    private double salary;

    // Constructor
```

```
public Employee(String employeeId, String name, String position, double salary) {
    this.employeeId = employeeId;
    this.name = name;
    this.position = position;
    this.salary = salary;
}

// Getters and Setters
public String getEmployeeId() {
    return employeeId;
}

public void setEmployeeId(String employeeId) {
    this.employeeId = employeeId;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getPosition() {
    return position;
}
```

```
    public void setPosition(String position) {
        this.position = position;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "employeeId='" + employeeId + '\'' +
                ", name='" + name + '\'' +
                ", position='" + position + '\'' +
                ", salary=" + salary +
                '}';
    }
}
```

3. Implementation

Employee Management System with Array:

java

```
public class EmployeeManager {
    private Employee[] employees;
    private int size; // Number of employees currently stored

    // Constructor
    public EmployeeManager(int capacity) {
        employees = new Employee[capacity];
        size = 0;
    }

    // Add an employee
    public void addEmployee(Employee employee) {
        if (size >= employees.length) {
            System.out.println("Array is full. Cannot add more employees.");
            return;
        }
        employees[size++] = employee;
    }

    // Search for an employee by employeeId
    public Employee searchEmployeeById(String employeeId) {
        for (int i = 0; i < size; i++) {
            if (employees[i].getEmployeeId().equals(employeeId)) {
                return employees[i];
            }
        }
        return null; // Not found
    }

    // Traverse and display all employees
    public void displayAllEmployees() {
        if (size == 0) {
            System.out.println("No employees to display.");
            return;
        }
        for (int i = 0; i < size; i++) {
            System.out.println(employees[i]);
        }
    }

    // Delete an employee by employeeId
    public void deleteEmployeeById(String employeeId) {
        int indexToDelete = -1;
        for (int i = 0; i < size; i++) {
            if (employees[i].getEmployeeId().equals(employeeId)) {
                indexToDelete = i;
                break;
            }
        }
        if (indexToDelete == -1) {
            System.out.println("Employee with ID " + employeeId + " not found.");
            return;
        }
```

```
        // Shift elements to the left
        for (int i = indexToDelete; i < size - 1; i++) {
            employees[i] = employees[i + 1];
        }
        employees[size - 1] = null; // Clear the last element
        size--;
    }
}
public class Main {
    public static void main(String[] args) {
        EmployeeManager manager = new EmployeeManager(5);

        // Adding employees
        manager.addEmployee(new Employee("E001", "John Doe", "Developer", 80000));
        manager.addEmployee(new Employee("E002", "Jane Smith", "Manager", 90000));
        manager.addEmployee(new Employee("E003", "Emily Johnson", "Analyst", 75000));

        // Display all employees
        System.out.println("All Employees:");
        manager.displayAllEmployees();

        // Search for an employee
        Employee emp = manager.searchEmployeeById("E002");
        if (emp != null) {
            System.out.println("Employee Found: " + emp);
        } else {
            System.out.println("Employee not found.");
        }

        // Delete an employee
        manager.deleteEmployeeById("E001");

        // Display all employees after deletion
        System.out.println("All Employees After Deletion:");
        manager.displayAllEmployees();
    }
}
```

Exercise 5: Task Management System

Create a Task class with attributes taskId, taskName, and status.

```
public class Task {
    private String taskId;
    private String taskName;
    private String status;

    // Constructor
    public Task(String taskId, String taskName, String status) {
        this.taskId = taskId;
        this.taskName = taskName;
        this.status = status;
    }
```

```
// Getters and Setters
public String getTaskId() {
    return taskId;
}

public void setTaskId(String taskId) {
    this.taskId = taskId;
}

public String getTaskName() {
    return taskName;
}

public void setTaskName(String taskName) {
    this.taskName = taskName;
}

public String getStatus() {
    return status;
}

public void setStatus(String status) {
    this.status = status;
}

@Override
```

```
    public String toString() {
        return "Task{" +
                "taskId='" + taskId + '\'' +
                ", taskName='" + taskName + '\'' +
                ", status='" + status + '\'' +
                '}';
    }
}
```

3. Implementation

Singly Linked List Implementation:

java

Copy code

```
public class SinglyLinkedList {

    // Node class for the linked list
    private static class Node {
        Task task;
        Node next;

        Node(Task task) {
            this.task = task;
            this.next = null;
        }
    }

    private Node head; // Head of the list
```

```
// Constructor
public SinglyLinkedList() {
    this.head = null;
}

// Add a task to the end of the list
public void addTask(Task task) {
    Node newNode = new Node(task);
    if (head == null) {
        head = newNode;
    } else {
        Node current = head;
        while (current.next != null) {
            current = current.next;
        }
        current.next = newNode;
    }
}

// Search for a task by taskId
public Task searchTaskById(String taskId) {
    Node current = head;
    while (current != null) {
        if (current.task.getTaskId().equals(taskId)) {
            return current.task;
        }
        current = current.next;
    }
    return null; // Not found
}

// Traverse and display all tasks
public void displayAllTasks() {
    Node current = head;
    if (current == null) {
        System.out.println("No tasks to display.");
        return;
    }
    while (current != null) {
        System.out.println(current.task);
        current = current.next;
    }
}

// Delete a task by taskId
public void deleteTaskById(String taskId) {
    if (head == null) {
        System.out.println("The list is empty.");
        return;
    }

    // If the head node itself is the task to be deleted
    if (head.task.getTaskId().equals(taskId)) {
            head = head.next;
            return;
        }

        // Search for the task to delete
        Node current = head;
        Node previous = null;
        while (current != null && !current.task.getTaskId().equals(taskId)) {
            previous = current;
            current = current.next;
        }

        // If the task was not found
        if (current == null) {
            System.out.println("Task with ID " + taskId + " not found.");
            return;
        }

        // Bypass the node to delete it
        previous.next = current.next;
    }
}
public class Main {
    public static void main(String[] args) {
        SinglyLinkedList taskList = new SinglyLinkedList();

        // Adding tasks
        taskList.addTask(new Task("T001", "Design Database", "In Progress"));
        taskList.addTask(new Task("T002", "Implement API", "Not Started"));
        taskList.addTask(new Task("T003", "Test Application", "Completed"));

        // Display all tasks
        System.out.println("All Tasks:");
        taskList.displayAllTasks();

        // Search for a task
        Task task = taskList.searchTaskById("T002");
        if (task != null) {
            System.out.println("Task Found: " + task);
        } else {
            System.out.println("Task not found.");
        }

        // Delete a task
        taskList.deleteTaskById("T001");

        // Display all tasks after deletion
        System.out.println("All Tasks After Deletion:");
        taskList.displayAllTasks();
    }
}
```

Exercise 6: Library Management System

Create a Book class with attributes bookId, title, and author.

```
public class Book {
    private String bookId;
    private String title;
    private String author;

    // Constructor
    public Book(String bookId, String title, String author) {
        this.bookId = bookId;
        this.title = title;
        this.author = author;
    }

    // Getters and Setters
    public String getBookId() {
        return bookId;
    }

    public void setBookId(String bookId) {
        this.bookId = bookId;
    }

    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    @Override
    public String toString() {
        return "Book{" +
                "bookId='" + bookId + '\'' +
                ", title='" + title + '\'' +
                ", author='" + author + '\'' +
                '}';
    }
}
```

3. Implementation

Linear Search Implementation:

java

Copy code

```java
import java.util.ArrayList;
import java.util.List;

public class LibraryManagementSystem {

    private List<Book> books;

    // Constructor
    public LibraryManagementSystem() {
        books = new ArrayList<>();
    }

    // Add a book
    public void addBook(Book book) {
        books.add(book);
    }

    // Linear search for a book by title
    public Book searchBookByTitleLinear(String title) {
        for (Book book : books) {
            if (book.getTitle().equalsIgnoreCase(title)) {
                return book;
            }
        }
        return null; // Book not found
    }
    // Binary search for a book by title (requires sorted list)
    public Book searchBookByTitleBinary(String title) {
        int left = 0;
        int right = books.size() - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;
            Book midBook = books.get(mid);

            int comparison = midBook.getTitle().compareToIgnoreCase(title);

            if (comparison == 0) {
                return midBook;
            } else if (comparison < 0) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return null; // Book not found
    }

    // Sort books by title (for binary search to work)
    public void sortBooksByTitle() {
        books.sort((b1, b2) -> b1.getTitle().compareToIgnoreCase(b2.getTitle()));
    }
}
```

4. Analysis

Comparison of Linear and Binary Search:

```
public class Main {
    public static void main(String[] args) {
        LibraryManagementSystem library = new LibraryManagementSystem();

        // Add books to the library
        library.addBook(new Book("B001", "The Catcher in the Rye", "J.D. Salinger"));
        library.addBook(new Book("B002", "To Kill a Mockingbird", "Harper Lee"));
        library.addBook(new Book("B003", "1984", "George Orwell"));
        library.addBook(new Book("B004", "The Great Gatsby", "F. Scott Fitzgerald"));

        // Linear search
        Book foundBookLinear = library.searchBookByTitleLinear("1984");
        if (foundBookLinear != null) {
            System.out.println("Linear Search - Book Found: " + foundBookLinear);
        } else {
            System.out.println("Linear Search - Book not found.");
        }

        // Sort books by title for binary search
        library.sortBooksByTitle();

        // Binary search
        Book foundBookBinary = library.searchBookByTitleBinary("1984");
        if (foundBookBinary != null) {
            System.out.println("Binary Search - Book Found: " + foundBookBinary);
        } else {
            System.out.println("Binary Search - Book not found.");
        }
    }
}
```

Exercise 7: Financial Forecasting

Java Code:

```
public class FinancialForecasting {

    // Method to calculate future value recursively
    public static double calculateFutureValue(double currentValue, double growthRate, int years) {
        // Base case: no years left
        if (years == 0) {
            return currentValue;
        }
        // Recursive case: calculate future value for one year less
        return calculateFutureValue(currentValue * (1 + growthRate), growthRate, years - 1);
    }

    public static void main(String[] args) {
        double initialAmount = 1000.0; // Initial investment amount
        double annualGrowthRate = 0.05; // 5% annual growth rate
        int numberOfYears = 10; // Number of years to forecast

        double futureValue = calculateFutureValue(initialAmount, annualGrowthRate, numberOfYears);
        System.out.println("Future Value after " + numberOfYears + " years: $" + String.format("%.2f", futureValue));
    }
}
```

Analysis

```
public class FinancialForecasting {

    // Method to calculate future value iteratively
    public static double calculateFutureValueIterative(double currentValue, double growthRate, int years) {
        for (int i = 0; i < years; i++) {
            currentValue *= (1 + growthRate);
        }
        return currentValue;
    }

    public static void main(String[] args) {
        double initialAmount = 1000.0; // Initial investment amount
        double annualGrowthRate = 0.05; // 5% annual growth rate
        int numberOfYears = 10; // Number of years to forecast

        // Using recursive approach
        double futureValueRecursive = calculateFutureValue(initialAmount, annualGrowthRate, numberOfYears);
        System.out.println("Future Value after " + numberOfYears + " years (Recursive): $" + String.format("%.2f", futureValueRecursive));

        // Using iterative approach
        double futureValueIterative = calculateFutureValueIterative(initialAmount, annualGrowthRate, numberOfYears);
        System.out.println("Future Value after " + numberOfYears + " years (Iterative): $" + String.format("%.2f", futureValueIterative));
    }
}
```