Guia Completo de Arrays e Listas em Java

Agosto 2025

Guia teórico e prático sobre Arrays e Listas em Java com exemplos e exercícios

Sumário

1 Introdução

Arrays e listas são estruturas de dados fundamentais em Java para armazenar e manipular coleções de elementos. Arrays oferecem uma estrutura estática de tamanho fixo, enquanto listas, parte do Java Collections Framework, proporcionam flexibilidade com tamanho dinâmico. Este guia cobre os conceitos teóricos, implementações práticas, exemplos e exercícios para dominar essas estruturas no Java 17.

1.1 Objetivo do Guia

Este documento tem como objetivo:

- Explicar os conceitos de arrays e listas em Java.
- Detalhar as principais classes de listas (`ArrayList`, `LinkedList`, etc.).
- Fornecer exemplos práticos de manipulação.
- Apresentar exercícios para consolidar o aprendizado.
- Comparar arrays e listas em termos de uso e desempenho.

2 Arrays

Arrays são estruturas de dados de tamanho fixo que armazenam elementos do mesmo tipo em posições contíguas na memória.

2.1 Características

- **Tamanho Fixo**: Definido na criação e imutável.
- **Acesso Rápido**: Usa índices para acesso direto ($O(1)$).
- **Tipo Homogêneo**: Todos os elementos devem ser do mesmo tipo.

2.2 Declaração e Inicialização

```java
public class ArrayExample {
    public static void main(String[] args) {
        // Declaração
        int[] numbers;

        // Inicialização com tamanho
        numbers = new int[5];

        // Inicialização com valores
        int[] primes = {2, 3, 5, 7, 11};

        // Array multidimensional
        int[][] matrix = {{1, 2}, {3, 4}};
    }
}
```

Listing 1: Declaração e Inicialização de Arrays

2.3 Acessando e Modificando Elementos

```java
public class ArrayManipulation {
    public static void main(String[] args) {
        int[] numbers = new int[3];

        // Atribuindo valores
        numbers[0] = 10;
        numbers[1] = 20;
        numbers[2] = 30;

        // Acessando valores
        System.out.println(numbers[1]); // Saída: 20

        // Iterando com for
        for (int i = 0; i < numbers.length; i++) {
            System.out.println(numbers[i]);
        }

        // Iterando com for-each
        for (int num : numbers) {
            System.out.println(num);
        }
    }
}
```

Listing 2: Acessando e Modificando Arrays

2.4 Exceções Comuns

- `ArrayIndexOutOfBoundsException`: Acesso a um índice inválido.
- `NullPointerException`: Tentativa de acessar um array não inicializado.

3 Listas

Listas são coleções dinâmicas que fazem parte do Java Collections Framework (`java.util`). Elas permitem adicionar, remover e modificar elementos em tempo de execução.

3.1 Principais Implementações

- **ArrayList**: Lista baseada em array, ideal para acesso rápido.
- **LinkedList**: Lista duplamente encadeada, eficiente para inserções/removals frequentes.
- **Vector**: Similar ao `ArrayList`, mas sincronizado (thread-safe).

- **CopyOnWriteArrayList**: Thread-safe, otimizado para leituras frequentes.

3.2 Interface List

A interface `java.util.List` define métodos comuns, como:

- `add(E e)`: Adiciona um elemento.
- `remove(int index)`: Remove um elemento por índice.
- `get(int index)`: Acessa um elemento por índice.
- `size()`: Retorna o tamanho da lista.

3.3 Exemplo com ArrayList

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

public class ArrayListExample {
    public static void main(String[] args) {
        List<String> names = new ArrayList<>();

        // Adicionando elementos
        names.add("Alice");
        names.add("Bob");
        names.add(1, "Charlie"); // Insere na posição 1

        // Acessando elementos
        System.out.println(names.get(0)); // Alice

        // Removendo elementos
        names.remove("Bob");

        // Iterando
        names.forEach(System.out::println);
    }
}
```

Listing 3: Exemplo com ArrayList

3.4 Exemplo com LinkedList

```java
import java.util.LinkedList;
import java.util.List;

public class LinkedListExample {
    public static void main(String[] args) {
        List<Integer> numbers = new LinkedList<>();

        // Adicionando no início e no final
        numbers.addFirst(1);
```

```
        numbers.addLast(2);

        // Removendo
        numbers.removeFirst();

        // Iterando
        for (Integer num : numbers) {
            System.out.println(num);
        }
    }
}
```

Listing 4: Exemplo com LinkedList

4 Comparação entre Arrays e Listas

Característica	Array	Lista
Tamanho	Fixo	Dinâmico
Desempenho de Acesso	$O(1)$	$O(1)$ (`ArrayList`), $O(n)$ (`LinkedList`)
Inserção/Remoção	Lento ($O(n)$)	Rápido (`LinkedList`, $O(1)$ para extremidades)
Tipo de Dados	Primitivos ou Objetos	Apenas Objetos
Thread-Safety	Não	`Vector`, `CopyOnWriteArrayList`

Table 1: Comparação entre Arrays e Listas

4.1 Quando Usar Cada Um

- **Arrays**: Para coleções de tamanho fixo com acesso rápido (ex.: matrizes, dados estáticos).
- **ArrayList**: Para listas dinâmicas com acesso frequente por índice.
- **LinkedList**: Para listas com muitas inserções/removals nas extremidades.
- **Vector**: Para cenários thread-safe com sincronização simples.

5 Exemplo Prático: Sistema de Gerenciamento de Tarefas

Abaixo, um sistema completo de gerenciamento de tarefas usando arrays e listas.

5.1 Classe Task

```
public class Task {
    private String description;
    private boolean completed;

    public Task(String description) {
        this.description = description;
```

```
        this.completed = false;
    }

    public String getDescription() { return description; }
    public boolean isCompleted() { return completed; }
    public void setCompleted(boolean completed) { this.completed
       = completed; }

    @Override
    public String toString() {
        return description + " [Completed: " + completed + "]";
    }
}
```

Listing 5: Classe Task.java

5.2 Gerenciamento com Array

```
public class TaskManagerArray {
    private Task[] tasks;
    private int size;

    public TaskManagerArray(int capacity) {
        tasks = new Task[capacity];
        size = 0;
    }

    public void addTask(Task task) {
        if (size < tasks.length) {
            tasks[size++] = task;
        }
    }

    public Task[] getCompletedTasks() {
        int count = 0;
        for (Task task : tasks) {
            if (task != null && task.isCompleted()) {
                count++;
            }
        }
        Task[] completed = new Task[count];
        int index = 0;
        for (Task task : tasks) {
            if (task != null && task.isCompleted()) {
                completed[index++] = task;
            }
        }
        return completed;
    }
}
```

Listing 6: Classe TaskManagerArray.java

5.3 Gerenciamento com ArrayList

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

public class TaskManagerList {
    private List<Task> tasks;

    public TaskManagerList() {
        tasks = new ArrayList<>();
    }

    public void addTask(Task task) {
        tasks.add(task);
    }

    public List<Task> getCompletedTasks() {
        return tasks.stream()
                    .filter(Task::isCompleted)
                    .collect(Collectors.toList());
    }
}
```

Listing 7: Classe TaskManagerList.java

5.4 Classe Main

```java
public class Main {
    public static void main(String[] args) {
        // Usando Array
        TaskManagerArray arrayManager = new TaskManagerArray(3);
        arrayManager.addTask(new Task("Write report"));
        arrayManager.addTask(new Task("Attend meeting"));
        arrayManager.tasks[0].setCompleted(true);
        for (Task task : arrayManager.getCompletedTasks()) {
            System.out.println(task);
        }

        // Usando ArrayList
        TaskManagerList listManager = new TaskManagerList();
        listManager.addTask(new Task("Code review"));
        listManager.addTask(new Task("Deploy application"));
        listManager.getTasks().get(0).setCompleted(true);
        listManager.getCompletedTasks().forEach(System.out::println);
    }
}
```

Listing 8: Classe Main.java

6 Exercícios

6.1 Exercício 1: Gerenciamento de Estoque

Crie um sistema de gerenciamento de estoque:

- Use um array para armazenar produtos com nome e quantidade.
- Implemente métodos para adicionar produtos e calcular o total de itens em estoque.
- Adapte o sistema para usar `ArrayList`.

Solução Parcial:

```
public class Product {
    private String name;
    private int quantity;

    public Product(String name, int quantity) {
        this.name = name;
        this.quantity = quantity;
    }

    public int getQuantity() { return quantity; }
}

public class Inventory {
    private Product[] products;
    private int size;

    public Inventory(int capacity) {
        products = new Product[capacity];
        size = 0;
    }

    public void addProduct(Product product) {
        if (size < products.length) {
            products[size++] = product;
        }
    }

    public int getTotalQuantity() {
        int total = 0;
        for (Product product : products) {
            if (product != null) {
                total += product.getQuantity();
            }
        }
        return total;
    }
}
```

Listing 9: Solução Parcial do Exercício 1

6.2 Exercício 2: Fila de Prioridade

Crie uma fila de prioridade usando `LinkedList`:

- Cada elemento tem uma prioridade (inteiro).
- Elementos com maior prioridade são processados primeiro.

6.3 Exercício 3: Análise de Dados

Implemente um programa que:

- Usa um array para armazenar números.
- Calcula média, máximo e mínimo.
- Adapte para usar `ArrayList` com streams.

7 Melhores Práticas

- **Escolha a Estrutura Certa**: Use arrays para dados fixos e listas para coleções dinâmicas.
- **Validação de Índices**: Sempre verifique limites ao acessar arrays.
- **Genéricos**: Use `List<T>` para garantir tipagem segura.
- **Streams**: Aproveite streams para manipulação eficiente de listas.
- **Evite Nulls**: Prefira listas vazias a null para coleções.

8 Conclusão

Arrays e listas são ferramentas essenciais em Java para manipulação de coleções. Arrays oferecem simplicidade e desempenho para tamanhos fixos, enquanto listas proporcionam flexibilidade e funcionalidades avançadas. Este guia cobriu teoria, exemplos práticos e exercícios para consolidar o aprendizado.

9 Referências

- https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/List.html
- Bloch, Joshua. *Effective Java*, 3rd Edition, 2018.
- https://www.oracle.com/java/technologies/javase/17/