Guia Completo de Arrays e Listas de Objetos em Java

Agosto 2025

Guia teórico e prático sobre Arrays e Listas de Objetos em Java com exemplos detalhados

Sumário

1 Introdução

Arrays e listas são estruturas fundamentais em Java para armazenar e manipular coleções de dados. Quando se trata de objetos, essas estruturas permitem organizar instâncias de classes, facilitando operações complexas como filtragem, ordenação e busca. Este guia explora arrays e listas de objetos em Java 17, com foco em sua implementação, manipulação, exemplos práticos e explicações detalhadas.

1.1 Objetivo do Guia

Este documento tem como objetivo:

- Explicar o funcionamento de arrays e listas para armazenar objetos.
- Detalhar as principais classes de listas (`ArrayList`, `LinkedList`).
- Fornecer exemplos práticos com objetos personalizados.
- Apresentar operações comuns, como iteração, filtragem e ordenação.
- Explorar boas práticas e exercícios para consolidar o aprendizado.

2 Arrays de Objetos

Arrays em Java são coleções de tamanho fixo que podem armazenar objetos de uma classe específica. Eles são alocados na memória de forma contígua, proporcionando acesso rápido por índice.

2.1 Características

- **Tamanho Fixo**: Definido na criação e imutável.
- **Acesso Rápido**: Acesso por índice em $O(1)$.
- **Tipo Específico**: Todos os elementos devem ser do mesmo tipo ou de uma superclasse.

2.2 Declaração e Inicialização

```
1  public class Person {
2      private String name;
3      private int age;
4
5      public Person(String name, int age) {
6          this.name = name;
7          this.age = age;
8      }
9
10     public String getName() { return name; }
11     public int getAge() { return age; }
12
13     @Override
```

```
    public String toString() {
        return name + " (" + age + ")";
    }
}

public class ArrayExample {
    public static void main(String[] args) {
        // Declaração
        Person[] people = new Person[3];

        // Inicialização
        people[0] = new Person("Alice", 25);
        people[1] = new Person("Bob", 30);
        people[2] = new Person("Charlie", 28);

        // Inicialização direta
        Person[] others = {
            new Person("Dave", 22),
            new Person("Eve", 27)
        };
    }
}
```

Listing 1: Declaração e Inicialização de Arrays de Objetos

2.3 Manipulação de Arrays

```
public class ArrayManipulation {
    public static void main(String[] args) {
        Person[] people = new Person[3];
        people[0] = new Person("Alice", 25);
        people[1] = new Person("Bob", 30);
        people[2] = new Person("Charlie", 28);

        // Acessando elementos
        System.out.println(people[1].getName()); // Bob

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

        // Iterando com for-each
        for (Person person : people) {
            System.out.println(person);
        }
    }
}
```

Listing 2: Manipulação de Arrays de Objetos

2.4 Exceções Comuns

- `ArrayIndexOutOfBoundsException`: Acesso a um índice fora dos limites.
- `NullPointerException`: Tentativa de acessar um elemento não inicializado.

3 Listas de Objetos

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

3.1 Principais Implementações

- **ArrayList**: Baseada em array, ideal para acesso rápido por índice ($O(1)$).
- **LinkedList**: Lista duplamente encadeada, eficiente para inserções e remoções ($O(1)$).
- **Vector**: Thread-safe, mas menos usada devido ao desempenho.
- **CopyOnWriteArrayList**: Thread-safe, otimizada para leituras frequentes.

3.2 Interface List

A interface `List` define métodos como:

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

3.3 Exemplo com ArrayList

```
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class ArrayListExample {
5      public static void main(String[] args) {
6          List<Person> people = new ArrayList<>();
7
8          // Adicionando objetos
9          people.add(new Person("Alice", 25));
10         people.add(new Person("Bob", 30));
11         people.add(1, new Person("Charlie", 28));
12
13         // Acessando
14         System.out.println(people.get(0).getName()); // Alice
15
16         // Removendo
17         people.removeIf(p -> p.getName().equals("Bob"));
```

```
18
19          // Iterando
20          people.forEach(System.out::println);
21      }
22  }
```

Listing 3: Exemplo com ArrayList de Objetos

3.4 Exemplo com LinkedList

```
1  import java.util.LinkedList;
2  import java.util.List;
3
4  public class LinkedListExample {
5      public static void main(String[] args) {
6          List<Person> people = new LinkedList<>();
7
8          // Adicionando no início e no final
9          people.addFirst(new Person("Alice", 25));
10         people.addLast(new Person("Bob", 30));
11
12         // Removendo
13         people.removeFirst();
14
15         // Iterando
16         people.forEach(System.out::println);
17     }
18 }
```

Listing 4: Exemplo com LinkedList de Objetos

4 Operações Avançadas

4.1 Filtragem com Streams

O Java 8 introduziu streams, que facilitam operações como filtragem e mapeamento.

```
1  import java.util.ArrayList;
2  import java.util.List;
3  import java.util.stream.Collectors;
4
5  public class StreamExample {
6      public static void main(String[] args) {
7          List<Person> people = new ArrayList<>();
8          people.add(new Person("Alice", 25));
9          people.add(new Person("Bob", 30));
10         people.add(new Person("Charlie", 28));
11
12         // Filtrando pessoas com idade > 26
13         List<Person> adults = people.stream()
14                                     .filter(p -> p.getAge() > 26)
```

```
                                        .collect(Collectors.toList());
        adults.forEach(System.out::println);
    }
}
```

Listing 5: Filtragem com Streams

4.2 Ordenação

Para ordenar objetos, implemente `Comparable` ou use `Comparator`.

```
public class Person implements Comparable<Person> {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() { return name; }
    public int getAge() { return age; }

    @Override
    public int compareTo(Person other) {
        return Integer.compare(this.age, other.age);
    }

    @Override
    public String toString() { return name + " (" + age + ")"; }
}

public class SortingExample {
    public static void main(String[] args) {
        List<Person> people = new ArrayList<>();
        people.add(new Person("Alice", 25));
        people.add(new Person("Bob", 30));
        people.add(new Person("Charlie", 28));

        // Ordenando por idade
        Collections.sort(people);
        people.forEach(System.out::println);
    }
}
```

Listing 6: Ordenação com Comparable

4.3 Ordenação com Comparator

```
import java.util.ArrayList;
import java.util.Comparator;
```

```java
import java.util.List;

public class ComparatorExample {
    public static void main(String[] args) {
        List<Person> people = new ArrayList<>();
        people.add(new Person("Alice", 25));
        people.add(new Person("Bob", 30));
        people.add(new Person("Charlie", 28));

        // Ordenando por nome
        people.sort(Comparator.comparing(Person::getName));
        people.forEach(System.out::println);
    }
}
```

Listing 7: Ordenação com Comparator

5 Exemplo Prático: Sistema de Gerenciamento de Livros

Abaixo, um sistema completo para gerenciar livros usando arrays e listas.

5.1 Classe Book

```java
public class Book {
    private String isbn;
    private String title;
    private String author;
    private double price;

    public Book(String isbn, String title, String author, double
       price) {
        this.isbn = isbn;
        this.title = title;
        this.author = author;
        this.price = price;
    }

    public String getIsbn() { return isbn; }
    public String getTitle() { return title; }
    public String getAuthor() { return author; }
    public double getPrice() { return price; }

    @Override
    public String toString() {
        return title + " by " + author + " (ISBN: " + isbn + ",
           $" + price + ")";
    }
}
```

Listing 8: Classe Book.java

5.2 Gerenciamento com Array

```java
public class BookManagerArray {
    private Book[] books;
    private int size;

    public BookManagerArray(int capacity) {
        books = new Book[capacity];
        size = 0;
    }

    public void addBook(Book book) {
        if (size < books.length) {
            books[size++] = book;
        }
    }

    public Book[] findBooksByAuthor(String author) {
        int count = 0;
        for (Book book : books) {
            if (book != null &&
               book.getAuthor().equalsIgnoreCase(author)) {
                count++;
            }
        }
        Book[] result = new Book[count];
        int index = 0;
        for (Book book : books) {
            if (book != null &&
               book.getAuthor().equalsIgnoreCase(author)) {
                result[index++] = book;
            }
        }
        return result;
    }
}
```

Listing 9: Classe BookManagerArray.java

5.3 Gerenciamento com ArrayList

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

public class BookManagerList {
    private List<Book> books;
```

```
7
8       public BookManagerList() {
9           books = new ArrayList<>();
10      }
11
12      public void addBook(Book book) {
13          books.add(book);
14      }
15
16      public List<Book> findBooksByAuthor(String author) {
17          return books.stream()
18                      .filter(b ->
                               b.getAuthor().equalsIgnoreCase(author))
19                      .collect(Collectors.toList());
20      }
21  }
```

Listing 10: Classe BookManagerList.java

5.4 Classe Main

```
1   public class Main {
2       public static void main(String[] args) {
3           // Usando Array
4           BookManagerArray arrayManager = new BookManagerArray(3);
5           arrayManager.addBook(new Book("123", "Java Basics",
               "John Doe", 29.99));
6           arrayManager.addBook(new Book("456", "OOP Design", "John
               Doe", 39.99));
7           arrayManager.addBook(new Book("789", "Advanced Java",
               "Jane Smith", 49.99));
8
9           Book[] doeBooks = arrayManager.findBooksByAuthor("John
               Doe");
10          for (Book book : doeBooks) {
11              System.out.println(book);
12          }
13
14          // Usando ArrayList
15          BookManagerList listManager = new BookManagerList();
16          listManager.addBook(new Book("123", "Java Basics", "John
               Doe", 29.99));
17          listManager.addBook(new Book("456", "OOP Design", "John
               Doe", 39.99));
18          listManager.addBook(new Book("789", "Advanced Java",
               "Jane Smith", 49.99));
19
20          listManager.findBooksByAuthor("John
               Doe").forEach(System.out::println);
21      }
22  }
```

Listing 11: Classe Main.java

6 Comparação entre Arrays e Listas de Objetos

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)
Flexibilidade	Limitada	Alta (métodos como `add`, `remove`)
Streams	Requer conversão	Nativo

Table 1: Comparação entre Arrays e Listas de Objetos

6.1 Quando Usar Cada Um

- **Arrays**: Para coleções fixas de objetos onde o tamanho é conhecido (ex.: lista fixa de produtos).
- **ArrayList**: Para coleções dinâmicas com acesso frequente por índice.
- **LinkedList**: Para operações frequentes de inserção/remoção nas extremidades.

7 Exercícios

7.1 Exercício 1: Gerenciamento de Estudantes

Crie um sistema para gerenciar estudantes:

- Crie uma classe `Student` com atributos `name` e `grade`.
- Use um array para armazenar estudantes e implemente um método para encontrar estudantes com nota acima de um limite.
- Adapte para `ArrayList` usando streams.

Solução Parcial:

```
1  public class Student {
2      private String name;
3      private double grade;
4
5      public Student(String name, double grade) {
6          this.name = name;
7          this.grade = grade;
8      }
9
10     public String getName() { return name; }
11     public double getGrade() { return grade; }
12
```

```
    @Override
    public String toString() { return name + " (" + grade + ")";
       }
}

public class StudentManagerArray {
    private Student[] students;
    private int size;

    public StudentManagerArray(int capacity) {
        students = new Student[capacity];
        size = 0;
    }

    public void addStudent(Student student) {
        if (size < students.length) {
            students[size++] = student;
        }
    }

    public Student[] findStudentsAboveGrade(double threshold) {
        int count = 0;
        for (Student student : students) {
            if (student != null && student.getGrade() >=
               threshold) {
                count++;
            }
        }
        Student[] result = new Student[count];
        int index = 0;
        for (Student student : students) {
            if (student != null && student.getGrade() >=
               threshold) {
                result[index++] = student;
            }
        }
        return result;
    }
}
```

Listing 12: Solução Parcial do Exercício 1

7.2 Exercício 2: Biblioteca de Músicas

Crie um sistema para gerenciar músicas:

- Crie uma classe `Song` com `title` e `artist`.
- Use `LinkedList` para adicionar músicas no início e no final.
- Implemente um método para listar músicas por artista.

7.3 Exercício 3: Ordenação de Produtos

Crie um sistema de produtos:

- Crie uma classe `Product` com `name` e `price`.
- Use `ArrayList` para ordenar produtos por preço usando `Comparator`.

8 Melhores Práticas

- **Use Genéricos**: Sempre especifique o tipo em listas (`List<Person>`) para segurança de tipo.
- **Valide Índices**: Evite `ArrayIndexOutOfBoundsException` com verificações.
- **Aproveite Streams**: Use streams para operações complexas em listas.
- **Evite Nulls**: Inicialize listas vazias e verifique `null` em arrays.
- **Implemente toString**: Facilite a depuração de objetos com `toString`.

9 Conclusão

Arrays e listas de objetos são ferramentas poderosas em Java para gerenciar coleções complexas. 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.

10 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/