

# **Capítulo 06:**

# **Estruturas de dados em Java**

Leonardo Moura Leitão

Marcelo Gonçalves Pinheiro Quirino

# Agenda

- Array
- Java Collection Framework
- Interface *Iterator*
- Interface *Comparable*
- Interface *Comparator*
- For que itera sobre coleções

# Array



Estático

- Array é uma coleção de **tamanho fixo**, de variáveis do **mesmo tipo**, referenciado por um nome comum.




Homogêneo

- Arrays podem ter **uma ou mais dimensões**.
- Em Java, arrays são **objetos**.

# Array

- Quando o array é criado, todos os elementos são inicializados para o **valor default** do tipo de dado.



int, long, double, etc – 0  
boolean – false  
objeto – null

- Se houver uma tentativa de acesso indevido aos índices do array, uma exceção será gerada: **ArrayIndexOutOfBoundsException**

# Array (Uma dimensão)

- Criação

```
int arrayDeInteiros[] = new int[10];
```

The diagram illustrates the components of the array creation code: `int` is the data type, `arrayDeInteiros` is the array name, `[]` indicates it's an array, `new` is the instantiation keyword, `int` is the element type, and `10` is the size.

- Nome do array
- Instanciado como qualquer objeto
- Define o tipo do array
- Indica que é um array
- Tamanho do array

- Acesso aos elementos do array

```
int e01 = arrayDeInteiros[0]; // Primeiro elemento
int e10 = arrayDeInteiros[9]; // Último elemento
int erro = arrayDeInteiros[10]; // Erro!!!
```

The diagram shows three examples of array access. The first two are valid: `arrayDeInteiros[0]` for the first element and `arrayDeInteiros[9]` for the last element. The third is an error: `arrayDeInteiros[10]` is out of bounds.

# Array (Uma dimensão)

- Array de tipos primitivos

```
int[] arrayDeInteiros = new int[10];  
  
for ( int i = 0; i < 10; i++ )  
{  
    arrayDeInteiros[i] = i + 1;  
}
```



# Array (Uma dimensão)

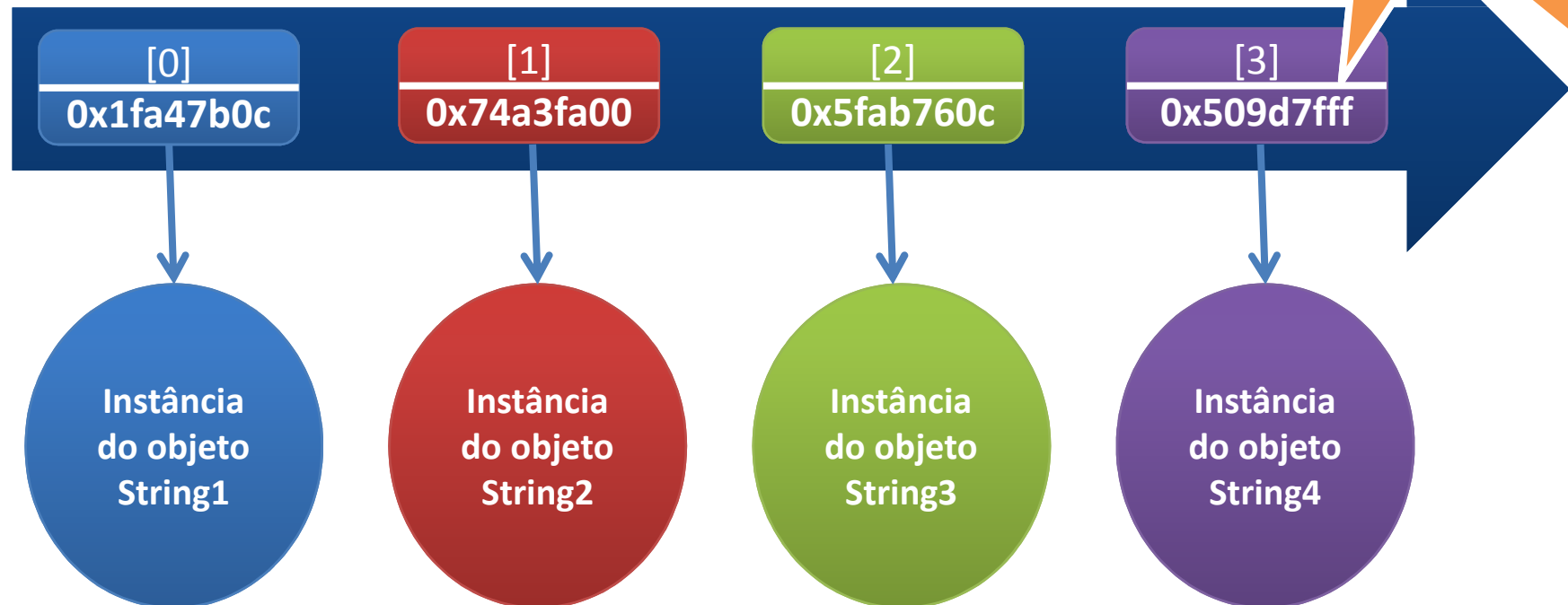
- Array de objetos

```
String[] arrayDeStrings = {"a", "b", "c", "d"};
```

Sintaxe alternativa

Iniciando com valores  
(não usa new)

Contém a  
referência p/ o  
objeto



# Array (Duas dimensões)

- Matriz simétrica

```
int tabela[][] = new int[3][4];  
for ( int y = 0; y < 3; y++ ) {  
    for ( int x = 0; x < 4; x++ ) {  
        tabela[y][x] = ( y * 4 ) + x + 1;  
    }  
}
```

[0][0] 1	[0][1] 2	[0][2] 3	[0][3] 4
[1][0] 5	[1][1] 6	[1][2] 7	[1][3] 8
[2][0] 9	[2][1] 10	[2][2] 11	[2][3] 12



# Array (Duas dimensões)

- Matriz assimétrica

```
int tabela[][] = new int[3][];  
tabela[0] = new int[7];  
tabela[1] = new int[2];  
tabela[2] = new int[5];  
for ( int y = 0, e = 1; y < tabela.length; y++ ) {  
    for ( int x = 0; x < tabela[y].length; x++ ) {  
        tabela[y][x] = e++;  
    }  
}
```

Obs.: Uma matriz nada mais é do que um array de array.

Usando o atributo **length** p/ controlar o laço **for**

[0][0] 1	[0][1] 2	[0][2] 3	[0][3] 4	[0][4] 5	[0][5] 6	[0][6] 7
[1][0] 8	[1][1] 9					
[2][0] 10	[2][1] 11	[2][2] 12	[2][3] 13	[2][4] 14		

# Array (N dimensões)

- Sintaxe

```
tipo nome[][]...[] = new tipo[tam1][tam2]...[tamN];
```

- Exemplos

```
int multidim[][][] = new int[4][10][3];
```

```
int[][][] multidim = new int[4][10][3];
```

```
int [][]multidim[] = new int[4][10][3];
```

```
int[] []multidim[] = new int[4][10][3];
```

# Java Collections Framework

- Uma coleção é simplesmente um objeto que **agrupa múltiplos elementos** em uma **única unidade**.
- As coleções são usadas para armazenar, recuperar, manipular e comunicar **dados agregados**.

Ex.: Lista telefônica

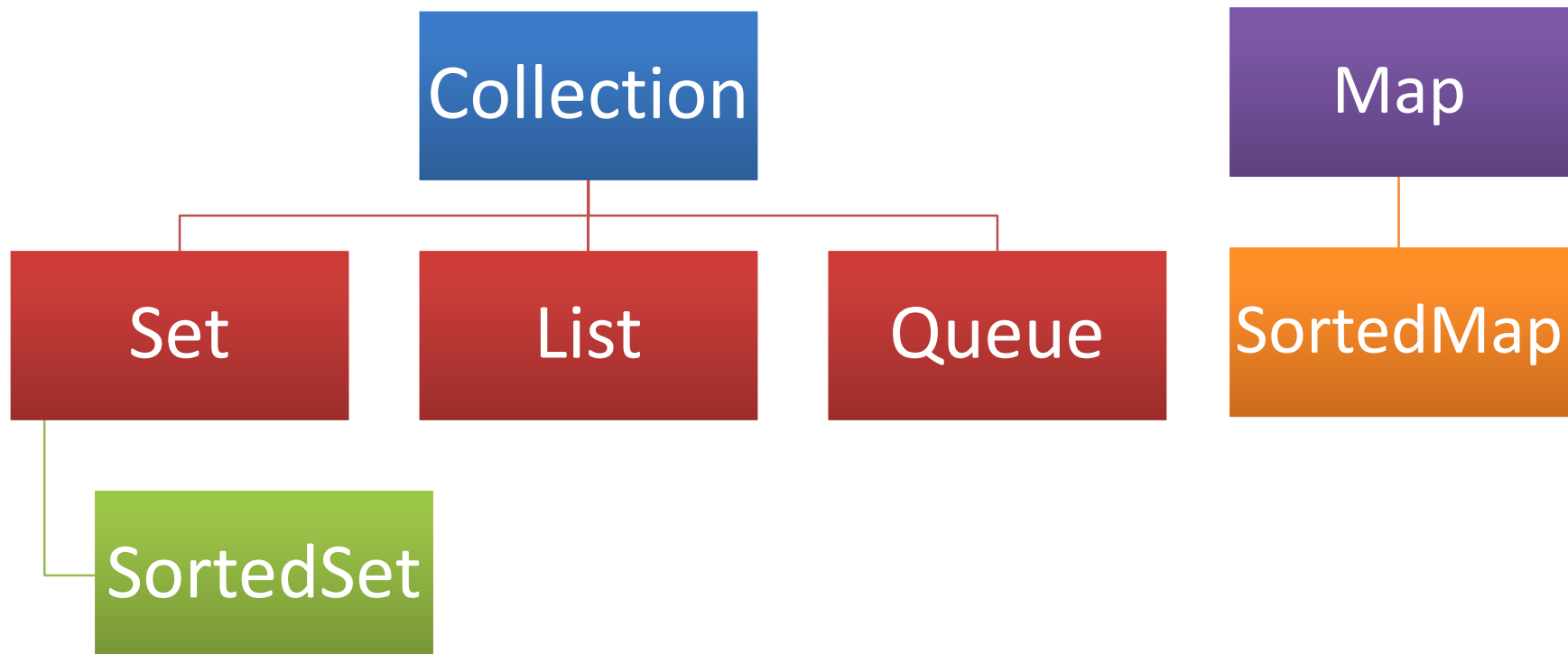
Nome	Telefone
Ana	32165498
Carlos	98765432

# Java Collections Framework

- Define uma **arquitetura** unificada para representar e manipular coleções.
- Dividida em três partes:
  - **Interfaces:** permite a manipulação independente dos detalhes de implementação.
  - **Implementação:** são as implementações concretas das interfaces.
  - **Algoritmos:** são métodos que executam processamentos úteis, como busca, ordenação, etc.

# Java Collections Framework

- As interfaces



# Java Collections Framework

Interface	Descrição
<b><i>Collection</i></b>	Uma interface básica que define as operações que todas as classes que trabalham com coleções de objetos tipicamente implementam.
<b><i>Set</i></b>	Estende a interface <i>Collection</i> para conjuntos que mantêm elementos únicos.
<b><i>SortedSet</i></b>	Aumenta a interface <i>Set</i> para conjuntos que mantêm seus elementos classificados de acordo com uma ordem.
<b><i>List</i></b>	Estende a interface <i>Collection</i> para listas que mantêm seus elementos numa seqüência, isto é, os elementos estão em ordem e podem ser duplicados.
<b><i>Queue</i></b>	Estende a interface <i>Collection</i> para filas, adicionando métodos para inserir, extrair, geralmente implementa FIFO.
<b><i>Map</i></b>	Uma interface básica que define as operações que classes que representam mapeamentos de chaves para valores tipicamente implementam.
<b><i>SortedMap</i></b>	Estende a interface <i>Map</i> para mapeamentos que mantêm seus pares chave-valor ordenados pela chave.

# Java Collections Framework

- **As implementações**

Interfaces	Implementações				
	Hash table	Resizable array	Tree	Linked list	Hash table + Linked list
<b>Set</b>	HashSet		TreeSet		LinkedHashSet
<b>List</b>		ArrayList		LinkedList	
<b>Queue</b>					
<b>Map</b>	HashMap		TreeMap		LinkedHashMap

# Interface *Iterator*

- Para navegar dentro de uma *Collection* e selecionar cada objeto em determinada seqüência.
  - Método **iterator()** (de *Collection*) retorna **Iterator**.

```
HashMap map = new HashMap();
map.put( "um", new Coisa( "um" ) );
map.put( "dois", new Coisa( "dois" ) );
map.put( "tres", new Coisa( "tres" ) );

Iterator it = map.values().iterator();
while ( it.hasNext() ) {
    Coisa c = (Coisa) it.next();
    System.out.println( c );
}
```



# Interface *Comparable*

- Interface usada para ordenação de objetos.
- Método a implementar

```
public int compareTo ( Object obj );
```

- Para implementar, retorne:
  - Um inteiro **menor que zero** se objeto atual for "menor" que o recebido como parâmetro;
  - Um inteiro **maior que zero** se objeto atual for "maior" que o recebido como parâmetro;
  - **Zero** se objetos forem iguais.

# Interface *Comparable*

```
public class Coisa implements Comparable {  
    public int id;  
    public Coisa ( int id ) {  
        this.id = id;  
    }  
    public int compareTo ( Object o ) {  
        Coisa outraCoisa = (Coisa) o;  
        if ( id > outraCoisa.id ) return 1;  
        else if ( id < outraCoisa.id ) return -1;  
        else return 0;  
    }  
}
```

```
Coisa coisas[] = { new Coisa( 4 ), new Coisa( 3 ) };  
Arrays.sort( coisas );
```

# Interface *Comparator*

- Interface usada para ordenação de objetos.
- Não exige que a classe do objeto a ser comparado implemente uma interface.
- Método a implementar

```
public int compare(Object o1, Object o2);
```

# Interface *Comparator*

```
public class ComparaCoisas implements Comparator
{
    public int compare ( Object o1, Object o2 )
    {
        Coisa c1 = (Coisa) o1;
        Coisa c2 = (Coisa) o2;

        if ( c1.id > c2.id ) return 1;
        else if ( c1.id < c2.id ) return -1;
        else return 0;
    }
}
```

```
Coisa coisas[] = { new Coisa( 4 ), new Coisa( 3 ) };
Arrays.sort( coisas, new ComparaCoisas() );
```

# For que itera sobre coleções

- A instrução **for** agora (a partir Java 5) aceita uma sintaxe alternativa.

```
for ( Objeto obj : coleção ) { ... }
```

- Pode ser lida como: “repita o bloco para cada Objeto obj da coleção”.
- Pode ser usado com **arrays** ou objetos iteráveis (que **implementem Iterable**).

Obs.: Collection  
estende Iterable