Programação Orientada a Objetos – Aula 07

Prof. Dr. Eduardo Takeo Ueda
eduardo.tueda@sp.senac.br

Arrays

- Estruturas de dados que consistem em itens de dados do mesmo tipo
- Permanecem com o mesmo tamanho depois de criados
- Array é um objeto

Arrays

Treinador.java

```
1  package senac.poo.aula6;
2
3  public class Treinador {
4
5      public static void main(String[] args) {
6
7          String nomeAtleta = "Bolt";
8
9          double[] marca = new double[5];
10         //double marca[] = new double[5];
11
12         marca[0] = 10.02;
13         marca[1] = 10.0;
14         marca[2] = 9.56;
15         marca[3] = 9.57;
16         marca[4] = 9.56;
17
18         for(int i = 0; i<5; i++)
19             System.out.println(nomeAtleta+" -> Tempo "+(i+1)+": "+marca[i]);
20     }
21 }
22
```

Arrays

Treinador.java

```java
package senac.poo.aula6;

public class Treinador {

    public static void main(String[] args) {

        String nomeAtleta = "Bolt";

        // double[] marca = {10.02, 10.0, 9.56, 9.57, 9.56};
        double marca[] = {10.02, 10.0, 9.56, 9.57, 9.56};

        for(int i = 0; i<5; i++)
            System.out.println(nomeAtleta+" -> Tempo "+(i+1)+": "+marca[i]);
    }
}

```

Arrays como Objetos

- Arrays são objetos, por isso é usado o comando new() para alocar espaço de armazenamento

- Atributo length

Treinador.java

```
1  package senac.poo.aula6;
2
3  public class Treinador {
4
5      public static void main(String[] args) {
6
7          String nomeAtleta = "Bolt";
8
9          // double[] marca = {10.02, 10.0, 9.56, 9.57, 9.56};
10         double marca[] = {10.02, 10.0, 9.56, 9.57, 9.56};
11
12         for(int i = 0; i<marca.length; i++)
13             System.out.println(nomeAtleta+" -> Tempo "+(i+1)+": "+marca[i]);
14     }
15 }
16
```

Arrays Bidimensionais

- Dimensão é o conjunto de valores necessários para localizar uma informação
- É possível definir *n* dimensões, porém, na prática, não é comum nem recomendável trabalhar com uma quantidade grande de dimensões
- É comum trabalharmos com apenas 1 (uma), e algumas vezes com 2 (duas), mas raramente com 3 (três) dimensões

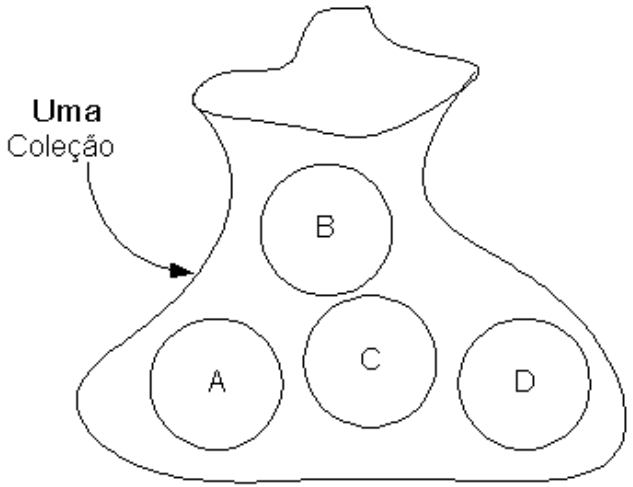
Arrays Bidimensionais

Xadrez.java

```
1  package senac.poo.aula6;
2
3  public class Xadrez {
4
5      public static void main(String[] args) {
6
7          // String[][] tabuleiro = new String[4][4];
8          String tabuleiro[][] = new String[4][4];
9
10         tabuleiro[0][0] = "Torre branca";
11         tabuleiro[0][1] = "Cavalo branco";
12         tabuleiro[0][3] = "Rainha branca";
13
14         System.out.println("Posição [0,0]: "+tabuleiro[0][0]);
15     }
16 }
17
```

Coleções

- A linguagem Java possui um conjunto de classes que servem para armazenar na memória vários objetos



Coleções

- Tais classes não possuem o inconveniente de termos que saber **a priori** a quantidade exata de elementos que iremos armazenar, como no caso de arrays

- Existem 3 tipos principais de objetos Java definidos para esse propósito
  - Set
  - List
  - Map

Coleções

- **Set** – representa a mesma ideia de conjuntos da matemática, ou seja, um grupo de objetos sem ordem definida, porém, únicos

- O principal representante dos Set é a classe HashSet

- Também temos TreeSet e LinkedHashSet

Coleções

TesteHashSet.java

```
1  package senac.poo.aula6;
2
3  import java.util.HashSet;
4  import java.util.Iterator;
5
6  public class TesteHashSet {
7
8      public static void main(String[] args) {
9
10         HashSet<String> itens = new HashSet<String>();
11         // HashSet itens = new HashSet();
12
13         itens.add("Suco");
14         itens.add("Chá");
15         itens.add("Café");
16         itens.add("Bolacha");
17         itens.add("Biscoito");
18
19         Iterator<String> i = itens.iterator();
20         // Iterator i = itens.iterator();
21
22         while( i.hasNext() )
23             System.out.println( i.next());
24     }
25 }
26
```

Coleções

- **List** – como o próprio nome sugere, representa uma lista de objetos, sendo que nela os objetos podem se repetir

- Nas listas definidas em Java, os objetos armazenados mantém a ordem em que foram adicionados

- Uma classe do tipo List bastante utilizada é a ArrayList

Coleções

TesteArrayList.java

```java
package senac.poo.aula6;

import java.util.ArrayList;

public class TesteArrayList {

    public static void main(String[] args) {

        ArrayList<String> nomes = new ArrayList<String>();
        // ArrayList nomes = new ArrayList();

        nomes.add("João");
        nomes.add("Maria");
        nomes.add("Josefina");
        nomes.add("Laura");
        nomes.add("José");

        for( String nome: nomes )
            System.out.println(nome);

        // for( int i=0; i<nomes.size(); i++ )
            // System.out.println(nomes.get(i));
    }
}
```

Coleções

- **Map** – Mapas são estruturas que relacionam um objeto a outro, por exemplo, um número de CEP ao nome de uma rua

- Podemos imaginar dois conjuntos, um de campos-chave e outro de objetos-valor que queremos armazenar

- Observe que para encontrar objetos é preciso localizá-los através de suas chaves

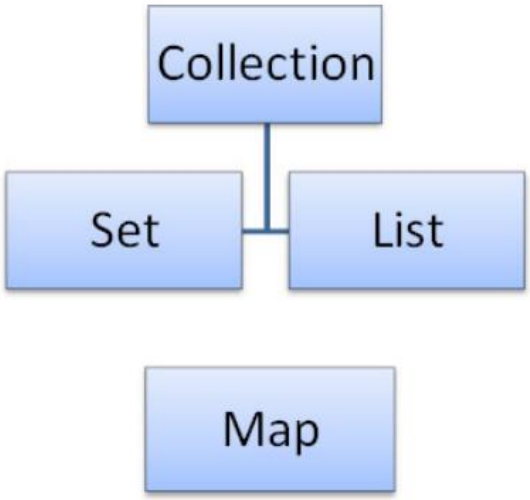
Coleções

TesteHashMap.java

```java
package senac.poo.aula6;

import java.util.HashMap;
import java.util.Set;

public class TesteHashMap {

    public static void main(String[] args) {

        HashMap<Integer, String> livros = new HashMap<Integer, String>();

        livros.put(1, "Volta ao mundo em 80 dias");
        livros.put(2, "Alice no país das maravilhas");
        livros.put(3, "O guia dos mochileiro das galáxias");

        Set<Integer> chaves = livros.keySet();
        for( Integer chave: chaves )
            if( chave != null )
                System.out.println(livros.get(chave));

        // for( int i=1; i<=livros.size(); i++ )
                // System.out.println(livros.get(i));
    }
}
```

Coleções

- Os Sets e Lists são do tipo Collection
- Apesar de Map não descender de Collection, é também uma classe que define métodos de armazenamento e recuperação de objetos



Coleções

- Lista básica de métodos da ***interface*** Collection

boolean add(Object)	Adiciona um elemento na coleção. Como algumas coleções não suportam elementos duplicados (exemplo: Sets), esses métodos retornam verdadeiro (true) ou falso (false) para indicar se a adição foi bem sucedida.
boolean remove(Object)	Remove determinado elemento da coleção. Se ele não fizer parte da coleção, retorna falso (false).
intsize()	Retorna a quantidade de elementos presentes na coleção.
boolean contains(Object)	Procura por um determinado objeto na coleção. Vale salientar: a comparação é feita pelo método equals().

Métodos e Classes Genéricas

- Seria interessante escrever um único método para ordenar elementos de um array de Integer, de um array de String ou de um array de qualquer tipo de elementos que possam ser comparados

- Também seria interessante escrever uma única classe Stack para ser utilizada como uma pilha de Integer, uma pilha de valores Double, uma pilha de String ou uma pilha de qualquer outro tipo

Métodos Sobrecarregados

Programa.java

```
1  package senac.poo.aula7;
2
3  public class Programa {
4
5      public static void main(String[] args) {
6
7          Integer[] integerArray = { 1, 2, 3, 4 };
8          Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
9          Character[] characterArray = { 'O', 'I' };
10
11         System.out.println("Array integerArray contém: ");
12         printArray(integerArray);
13
14         System.out.println("\nArray doubleArray contém: ");
15         printArray(doubleArray);
16
17         System.out.println("\nArray characterArray contém: ");
18         printArray(characterArray);
19     }
```

Métodos Sobrecarregados

```
19      }
20      public static void printArray( Integer[] inputArray) {
21          for ( Integer elemento : inputArray)
22              System.out.printf("%s ", elemento);
23      }
24
25      public static void printArray( Double[] inputArray ) {
26          for( Double elemento : inputArray )
27              System.out.printf("%s ", elemento);
28      }
29      public static void printArray( Character[] inputArray ) {
30              for( Character elemento : inputArray )
31                  System.out.printf("%s ", elemento);
32      }
33  }
```

Métodos Sobrecarregados

```
19      }
20      public static void printArray( Integer[] inputArray) {
21          for ( Integer elemento : inputArray)
22              System.out.printf("%s ", elemento);
23      }
24
25      public static void printArray( Double[] inputArray ) {
26          for( Double elemento : inputArray )
27              System.out.printf("%s ", elemento);
28      }
29      public static void printArray( Character[] inputArray ) {
30              for( Character elemento : inputArray )
31                  System.out.printf("%s ", elemento);
32      }
33  }
```

Métodos Genéricos

- Esses métodos sobrecarregados podem ser escritos de maneira mais compacta com um método genérico

- Utilizando um tipo genérico, é possível declarar um método printArray que pode exibir os elementos de qualquer array de objetos

Métodos Genéricos

Programa.java

```
1  package senac.poo.aula7;
2
3  public class Programa {
4
5      public static void main(String[] args) {
6
7          Integer[] integerArray = { 1, 2, 3, 4 };
8          Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
9          Character[] characterArray = { 'O', 'I' };
10
11         System.out.println("Array integerArray contém: ");
12         printArray(integerArray);
13
14         System.out.println("\nArray doubleArray contém: ");
15         printArray(doubleArray);
16
17         System.out.println("\nArray characterArray contém: ");
18         printArray(characterArray);
19     }
20     public static <T> void printArray( T[] inputArray) {
21         for ( T elemento : inputArray)
22             System.out.printf("%s ", elemento);
23     }
24 }
```

Métodos Genéricos

```
19      }
20      public static <T> void printArray( T[] inputArray) {
21          for ( T elemento : inputArray)
22              System.out.printf("%s ", elemento);
23      }
24  }
```

Métodos Genéricos

- O corpo de um método genérico é declarado como o de qualquer outro método

- Todas as declarações de métodos genéricos têm uma seção de parâmetros de tipos, delimitada por colchetes angulares

  public static <T> void printArray(T[] inputArray)

- A seção de parâmetros de tipos pode conter um ou mais parâmetros de tipos, separados por vírgulas

Métodos Genéricos

- Situação: Considere a tarefa de implementar um método genérico chamado max para retornar o maior elemento dentre três elementos do mesmo tipo

- Problema: O operador relacional **>** (maior que) não pode ser usado com objetos

- Solução: É possível comparar dois objetos da mesma classe se essa classe implementa a interface genérica Comparable <T>

Métodos Genéricos

Programa.java

```
1  package senac.poo.aula7;
2
3  public class Programa {
4
5      public static void main(String[] args) {
6
7          System.out.println("Maior: "+ max(1, 5, 3));
8          System.out.println("Maior: "+ max("José", "Maria", "Joaquim"));
9      }
10     public static <T extends Comparable <T>> T max( T x, T y, T z ) {
11
12         T m = x;
13         if(y.compareTo(m) > 0)
14             m = y;
15         if(z.compareTo(m) > 0)
16             m = z;
17         return m;
18     }
19 }
```

Métodos Genéricos

Programa.java

```
1  package senac.poo.aula7;
2
3  public class Programa {
4
5      public static void main(String[] args) {
6
7          System.out.println("Maior: "+ max(1, 5, 3));
8          System.out.println("Maior: "+ max("José", "Maria", "Joaquim"));
9      }
10     public static <T extends Comparable <T>> T max( T x, T y, T z ) {
11
12         T m = x;
13         if(y.compareTo(m) > 0)
14             m = y;
15         if(z.compareTo(m) > 0)
16             m = z;
17         return m;
18     }
19 }
```

Classes Genéricas

- O conceito de uma estrutura de dados, como uma pilha (stack), pode ser entendido independentemente do tipo que ela manipula, afinal uma pilha de Double, Integer ou Char funcionam da mesma maneira

- Classes genéricas fornecem uma maneira de descrever o conceito de uma pilha (ou de qualquer outra classe) de uma maneira independente do tipo

Classes Genéricas

- Classes genéricas são conhecidas como classes parametrizadas porque aceitam um ou mais parâmetros

- A declaração de uma classe genérica se parece com a declaração de uma não-genérica, exceto que o nome da classe é seguido por uma seção de parâmetros de tipo

  public class Stack <E>

Classes Genéricas

Stack.java

```
1  package senac.poo.aula7;
2
3  public class Stack <E> {
4
5      private final int size; //número de elementos da pilha
6      private int top; //localização do elemento superior
7      private E[] elementos; // array que armazena os elementos da pilha
8
9      public Stack( int s) {
10         size = s > 0 ? s : 10;
11         top = -1; //Stack inicialmente vazia
12         elementos = ( E[] ) new Object[ size ];
13     }
14
```

Classes Genéricas

```
14
15      public int push( E elemento) {
16          if( top == size -1) {
17              System.out.println("Pilha cheia!");
18              return 0;
19          }
20          else {
21              System.out.println("Inserindo " + elemento);
22              elementos[ ++top ] = elemento;
23              return 1;
24          }
25      }
26
27      public E pop() {
28          if( top == -1) {
29              System.out.println("Pilha vazia!");
30              return null;
31          }
32          return elementos[ top --];
33      }
34  }
```

Classes Genéricas

Programa.java

```
1  package senac.poo.aula7;
2
3  public class Programa {
4
5      public static void main(String[] args) {
6          Stack<Double> doubleStack = new Stack<Double>(3);
7          Stack<String> stringStack = new Stack<String>(5);
8
9          doubleStack.push(1.1);
10         doubleStack.push(1.5);
11         doubleStack.push(1.3);
12         doubleStack.push(1.7);
13
14         Double d = doubleStack.pop();
15         if (d != null)
16             System.out.println("Elemento removido: " + d);
17         else
18             System.out.println(d);
19
20         stringStack.push("Oi");
21         stringStack.push("Pessoal");
22         // ...
23     }
24 }
```