

SCC0204 - Programação Orientada a Objetos

Java Generics

Prof. Jose Fernando Rodrigues **Junior**

<http://www.icmc.usp.br/~junio>

junio@icmc.usp.br

**INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
- USP -**

Introdução

- A partir da versão 5.0, a linguagem Java passou a suportar classes parametrizadas, ou Generics, um recurso semelhante às templates de C++
- Um exemplo é a classe `ArrayList`
 - Quais tipos de objetos ela deve armazenar?

Exemplo de uma classe simples com parâmetro T

Display 14.4 A Class Definition with a Type Parameter

```
1 public class Sample<T>
2 {
3     private T data;
4
5     public void setData(T newData)
6     {
7         data = newData;
8
9     public T getData()
10    {
11        return data;
12    }
```

T is a parameter for a type.

Exemplo de uma classe simples com parâmetro T

- A classe Sample é denominada “classe genérica” ou “parametrizada”
 - ❑ Os parâmetros devem ser incluídos dentro de **colchetes angulares** após o nome da classe
 - ❑ Qualquer palavra não-chave pode ser usada, mas **por convenção**, o parâmetro começa com uma letra maiúscula
 - ❑ Os parâmetros, então, são usados ao longo da classe

Sintaxe

- A sintaxe em colchetes angulares não é usada dentro do corpo da classe

```
public Sample<T>()
```

- Para a definição acima, por exemplo, o construtor seria:

```
public Sample(T umDado)
```

- No entanto, quando uma instância de uma classe parametriza é criada, a sintaxe é necessária

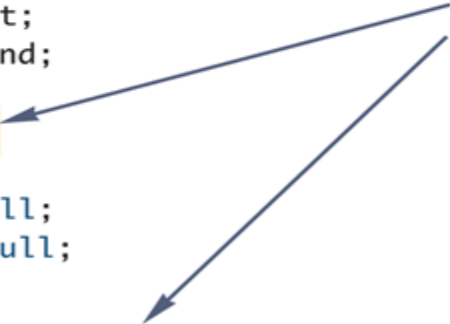
```
Pair<String> pair = new Pair<String>("Mario", "Prado");
```

Exemplo - Pair

Display 14.5 A Generic Ordered Pair Class

```
1  public class Pair<T>
2  {
3      private T first;
4      private T second;
5
6      public Pair()
7      {
8          first = null;
9          second = null;
10
11      public Pair(T firstItem, T secondItem)
12      {
13          first = firstItem;
14          second = secondItem;
15
16      public void setFirst(T newFirst)
17      {
18          first = newFirst;
19
20      public void setSecond(T newSecond)
21      {
22          second = newSecond;
23
24      public T getFirst()
25      {
26          return first;
27      }
```

Constructor headings do not include the type parameter in angular brackets.



➔ Exemplo NetBeans - Pair

(continued)

Exemplo - Pair

Display 14.5 A Generic Ordered Pair Class

```
27     public T getSecond()
28     {
29         return second;
30     }

31     public String toString()
32     {
33         return ( "first: " + first.toString() + "\n"
34                 + "second: " + second.toString() );
35     }
36
37     public boolean equals(Object otherObject)
38     {
39         if (otherObject == null)
40             return false;
41         else if (getClass() != otherObject.getClass())
42             return false;
43         else
44         {
45             Pair<T> otherPair = (Pair<T>)otherObject;
46             return (first.equals(otherPair.first)
47                     && second.equals(otherPair.second));
48         }
49     }
50 }
```

Exemplo - Pair

Display 14.6 Using Our Ordered Pair Class

```
1  import java.util.Scanner;

2  public class GenericPairDemo
3  {
4      public static void main(String[] args)
5      {
6          Pair<String> secretPair =
7              new Pair<String>("Happy", "Day");
8
9          Scanner keyboard = new Scanner(System.in);
10         System.out.println("Enter two words:");
11         String word1 = keyboard.next();
12         String word2 = keyboard.next();
13         Pair<String> inputPair =
14             new Pair<String>(word1, word2);
15
16         if (inputPair.equals(secretPair))
17         {
18             System.out.println("You guessed the secret words");
19             System.out.println("in the correct order!");
20         }
21         else
22         {
23             System.out.println("You guessed incorrectly.");
24             System.out.println("You guessed");
25             System.out.println(inputPair);
26             System.out.println("The secret words are");
27             System.out.println(secretPair);
28         }
29     }
```


- Classes parametrizadas **não podem receber tipos primitivos** como parâmetros; como int e double
- Para o uso destes tipos, a compilação Java provê automaticamente **tipos embutidos**
 - ❑ int → new Integer
 - ❑ double → new Double
 - ❑ ...

Tipos primitivos embutidos - exemplo

Display 14.7 Using Our Ordered Pair Class and Automatic Boxing

```
1  import java.util.Scanner;

2  public class GenericPairDemo2
3  {
4      public static void main(String[] args)
5      {
6          Pair<Integer> secretPair =
7              new Pair<Integer>(42, 24);
8
9          Scanner keyboard = new Scanner(System.in);
10         System.out.println("Enter two numbers:");
11         int n1 = keyboard.nextInt();
12         int n2 = keyboard.nextInt();
13         Pair<Integer> inputPair =
14             new Pair<Integer>(n1, n2);
15
16         if (inputPair.equals(secretPair))
17         {
18             System.out.println("You guessed the secret numbers");
19             System.out.println("in the correct order!");
20         }
21         else
22         {
23             System.out.println("You guessed incorrectly.");
24             System.out.println("You guessed");
25             System.out.println(inputPair);
26             System.out.println("The secret numbers are");
27             System.out.println(secretPair);
28         }
29     }
}
```

*Automatic boxing allows you to use an **int** argument for an **Integer** parameter.*

Múltiplos parâmetros

- Uma classe genérica pode ter qualquer número de parâmetros
 - A sintaxe é a mesma, basta separar os parâmetros por vírgulas

Múltiplos parâmetros - exemplo

Display 14.8 Multiple Type Parameters

```
1  public class TwoTypePair<T1, T2>
2  {
3      private T1 first;
4      private T2 second;

5      public TwoTypePair()
6      {
7          first = null;
8          second = null;
9      }

10     public TwoTypePair(T1 firstItem, T2 secondItem)
11     {
12         first = firstItem;
13         second = secondItem;
14     }

15     public void setFirst(T1 newFirst)
16     {
17         first = newFirst;
18     }

19     public void setSecond(T2 newSecond)
20     {
21         second = newSecond;
22     }

23     public T1 getFirst()
24     {
25         return first;
26     }
```

→ Exemplo NetBeans - PairDeDoisTipos

(continued)

Múltiplos parâmetros

Display 14.8 Multiple Type Parameters

```
27     public T2 getSecond()
28     {
29         return second;
30     }

31     public String toString()
32     {
33         return ( "first: " + first.toString() + "\n"
34                 + "second: " + second.toString() );
35     }
36
37     public boolean equals(Object otherObject)
38     {
39         if (otherObject == null)
40             return false;
41         else if (getClass() != otherObject.getClass())
42             return false;
43         else
44         {
45             TwoTypePair<T1, T2> otherPair =
46                 (TwoTypePair<T1, T2>)otherObject;
47             return (first.equals(otherPair.first)
48                     && second.equals(otherPair.second));
49         }
50     }
51 }
```

The first equals is the equals of the type T1. The second equals is the equals of the type T2.

Múltiplos parâmetros

Display 14.9 Using a Generic Class with Two Type Parameters

```
1  import java.util.Scanner;

2  public class TwoTypePairDemo
3  {
4      public static void main(String[] args)
5      {
6          TwoTypePair<String, Integer> rating =
7              new TwoTypePair<String, Integer>("The Car Guys", 8);

8          Scanner keyboard = new Scanner(System.in);
9          System.out.println(
10              "Our current rating for " + rating.getFirst());
11          System.out.println(" is " + rating.getSecond());

12          System.out.println("How would you rate them?");
13          int score = keyboard.nextInt();
14          rating.setSecond(score);
15          System.out.println(
16              "Our new rating for " + rating.getFirst());
17          System.out.println(" is " + rating.getSecond());
18      }
19  }
```

Limitantes para os parâmetros

- Para se trabalhar com um tipo “desconhecido” faz sentido **saber do que esse tipo é capaz**
- É possível **definir limitantes** para quais tipos podem ser fornecidos a uma classe parametrizada
- Pode-se **exigir** que um determinado tipo herde de uma determinada classe ou implemente uma dada interface, por exemplo:
 - Para garantir que uma classe genérica seja serializável, pode-se requerer a interface Serializable:

```
public class ClasseExemplo<T extends Serializable>
```

Obs.: nesta sintaxe não se usa a palavra chave “implements”

Exemplo – interface Comparable

Display 14.10 A Bounded Type Parameter

```
1  public class Pair<T extends Comparable>
2  {
3      private T first;
4      private T second;
5
6      public T max()
7      {
8          if (first.compareTo(second) <= 0)
9              return first;
10         else
11             return second;
12     }
```

```
12 }
```

Exemplo NetBeans → PairComparable

Métodos Genéricos

- É possível trabalhar com métodos genéricos, **independentemente** da classe ser genérica ou não
 - Mesmo que a classe não receba parâmetro, seus métodos podem receber
 - Mesmo em uma classe que recebe parâmetros, **os parâmetros dos métodos podem ser diferentes dos da classe**

Métodos Genéricos

- Sintaxe da definição de um método genérico

```
public <U> void MetodoGenerico (U a)
```

- Sintaxe do uso de um método genérico

```
String c = "teste";  
UmaClasse umaClasse = new UmaClasse();  
umaClasse.<String>MetodoGenerico(c);
```

Exemplo NetBeans → PairComparable

Herança com classes genéricas

- Classes genéricas podem ser usadas em **herança** como qualquer outra, podendo herdar de classes não genéricas ou genéricas

Herança com classes genéricas

Display 14.11 A Derived Generic Class

```
1  public class UnorderedPair<T> extends Pair<T>
2  {
3      public UnorderedPair()
4      {
5          setFirst(null);
6          setSecond(null);
7      }
8
9      public UnorderedPair(T firstItem, T secondItem)
10     {
11         setFirst(firstItem);
12         setSecond(secondItem);
13     }
14     public boolean equals(Object otherObject)
15     {
16         if (otherObject == null)
17             return false;
18         else if (getClass() != otherObject.getClass())
19             return false;
20         else
21         {
22             UnorderedPair<T> otherPair =
23                 (UnorderedPair<T>)otherObject;
24             return (getFirst().equals(otherPair.getFirst())
25                 && getSecond().equals(otherPair.getSecond()))
26                 ||
27                 (getFirst().equals(otherPair.getSecond())
28                 && getSecond().equals(otherPair.getFirst()));
29         }
30     }
31 }
```

Herança com classes genéricas

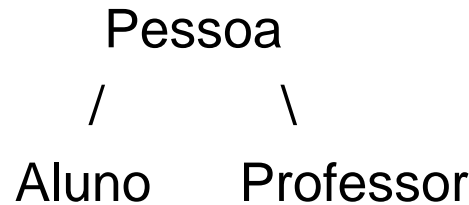
Display 14.12 Using UnorderedPair

```
1  public class UnorderedPairDemo
2  {
3      public static void main(String[] args)
4      {
5          UnorderedPair<String> p1 =
6              new UnorderedPair<String>("peanuts", "beer");
7          UnorderedPair<String> p2 =
8              new UnorderedPair<String>("beer", "peanuts");
9          if (p1.equals(p2))
10         {
11             System.out.println(p1.getFirst() + " and " +
12                                 p1.getSecond() + " is the same as");
13             System.out.println(p2.getFirst() + " and "
14                                 + p2.getSecond());
15         }
16     }
17 }
```

Wildcards – como passar
um tipo genérico como
parametro?

Wildcards

- Suponha que você possui uma hierarquia de classes



- Você possui coleções ArrayList Alunos
- E você deseja escrever um método que imprime o conteúdo desta coleção, seja lá qual for o conteúdo das coleções
- Pode-se pensar na seguinte solução

```
public void printCollection(ArrayList<Pessoa> umaColecao){  
    for(Pessoa o : umaColecao)  
        System.out.println(o.getNome());  
}
```

- Funciona?

Wildcards

Importante noção:

Dada uma classe genérica $G<T>$

Dadas duas classes quaisquer A e B

→ $G<A>$ não tem qualquer relação com G; **são classes absolutamente diferentes, mesmo que sejam relacionadas por herança**

Wildcards

- Resposta: **não funciona para alunos (nem para professores)**
- Apesar da hierarquia de classes, para o compilador **não há relação nenhuma entre as diferentes definições:**

`ArrayList<Pessoa> ≠ ArrayList<Aluno> ≠ ArrayList<Professor>`

- Desta maneira, o método só compila se receber **exatamente** um `ArrayList<Pessoa>`
- Como proceder então?
 - ➔ **Usar WildCard**
- O código ficaria então:

```
public void printCollection(ArrayList<?> umaColecao){  
    for(Pessoa o : umaColecao)  
        System.out.println(o.getNome());  
}
```

Wildcards

- O código ficaria então:

```
public void printCollection(ArrayList<?> umaColecao){  
    for(Pessoa o : umaColecao)  
        System.out.println(o.getNome());  
}
```

- Ok, mas e se alguém passasse alguma coisa que não é Pessoa? Por exemplo, ArrayList<Double>?

- Haveria problema de execução, pois Double não possui um método getNome

- Solução, limitar o wildcard; ficaria assim:

```
public void printCollection(ArrayList<? extends Pessoa> umaColecao){  
    for(Pessoa o : umaColecao)  
        System.out.println(o.getNome());  
}
```

→ Seriam aceitos ArrayList<Pessoa>, ArrayList<Aluno> e ArrayList<Professor>

Wildcards

- Solução, limitar o wildcard, ficaria assim:

```
public void printCollection(ArrayList<? extends Pessoa> umaColecao){  
    for(Pessoa o : umaColecao)  
        System.out.println(o.getNome());  
}
```

→ Seriam aceitos `ArrayList<Pessoa>`, `ArrayList<Aluno>` e `ArrayList<Professor>`

→ “`? extends`” é denominado upper bound wildcard

- Mas, e se por alguma razão fosse desejado que apenas `Professor` e `Pessoa` pudessem ser aceitos?

- A solução poderia ficar assim

```
public void printCollection(ArrayList<? super Professor> umaColecao){  
    for(Pessoa o : umaColecao)  
        System.out.println(o.getNome());  
}
```

→ Seriam aceitos `ArrayList<Professor>`, `ArrayList<Pessoa>`

→ “`? super`” é denominado lower bound wildcard