

# **Java Generics**

# **Técnicas de Programação**

**Jose Macedo**

# Objetivos

- Entender o objetivo de programacao generica
- Implementar classes e metodos genericos
- Conhecer as limitacoes de programacao generica em Java
- Entender o relacionamento de tipos genericos e heranca

# Classes Parametrizadas e Genericas

- A classe `ArrayList` é uma classe parametrizada
- Ela tem um parametro, chamado de `Base_Type`, o qual pode ser substituido por qualquer Tipo Abstrato de Dado (TAD) para obter uma classe para `ArrayLists` com este tipo de dado
- Iniciada na versao 5.0, permite a definicao de classes parametrizadas para TAD
- Classes que tem parametros de tipos sao chamadas de classes parametrizadas ou definicoes genericas, ou simplesmente, genericas (do ingles, generics)

## Generics (cont)

- A definição de classe com um parametro de tipo é armazenada em um arquivo e compilada com qualquer outra classe
- Uma vez que a classe parametrizada é compilada, ela pode ser usada como qualquer outra classe.
  - No entanto, o tipo de classe "plugado" ao parametro de tipo deve ser especificado antes de ser usado pelo programa.
  - Fazendo isso, diz-se que a classe generica foi instanciada.

```
Sample<String> object = new Sample<String>();
```

# Uma definicao de classe com parametro de tipo

**Display 14.4**     **A Class Definition with a Type Parameter**

---

```
1  public class Sample<T>
2  {
3      private T data;

4      public void setData(T newData)
5      {
6          data = newData;
7      }

8      public T getData()
9      {
10         return data;
11     }
12 }
```

*T is a parameter for a type.*

# Uma definicao de classe com parametro de tipo

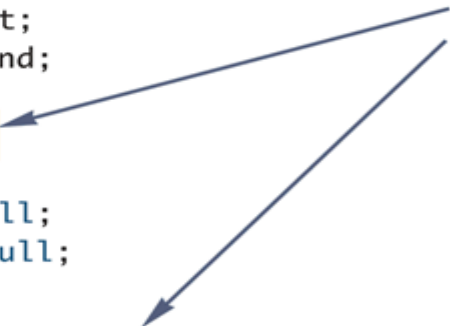
- Uma classe que é definida com um parametro para um tipo é chamada de classe generica ou uma classe parametrizada
  - O tipo de parametro é incluído entre sinal de maior e menor depois do nome da classe no cabeçalho da definicao.
  - Qualquer identificador, que não seja palavra-reservada, pode ser usado como parametro de tipo, mas por convencao, o parametro começa com um letra maiuscula.
  - O parametro de tipo pode ser usado como outros tipos usados na definicao de uma classe.

# Definicao de Classe Generica: Um Exemplo

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;
```

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



(continued)

# Definicao de Classe Generica: Um Exemplo

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 }
```



# Uso de Classe Generica: Um Exemplo

## 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     }
```

# Uso de Classe Generica: Um Exemplo

## Saida do Programa:

Display 14.6 Using Our Ordered Pair Class

---

### SAMPLE DIALOGUE

Enter two words:

`two words`

You guessed incorrectly.

You guessed

first: two

second: words

The secret words are

first: Happy

second: Day

# O Construtor da classe generica NAO pode possuir parametro de tipo !!!

- Embora a classe parametrizada tenha um parametro de tipo, este NAO pode usado na definicao do construtor da classe, como no exemplo abaixo:

```
public Pair<T>() //NAO PODE !
```

- Um construtor pode usar o parametro de tipo com o tipo para seus parametros, porém neste caso, o simbolo maior-menor nao devem ser usados :

```
public Pair(T first, T second)
```

- No entanto, quando uma classe generica é instanciada, o simbolo maior-menor devem ser usados:

```
Pair<String> pair = new Pair<String>("Happy", "Day");
```

## Um tipo primitivo NAO pode ser plugado em um parametro de tipo!!!

- O tipo a ser plugado em um parametro de tipo tem que ser sempre um tipo classe:
  - Ele nao pode ser um tipo primitivo como: `int`, `double`, ou `char`
  - Porém, como Java tem mecanismo de autoboxing isso nao é uma grande restricao.
  - Nota: Tipo de Objeto podem incluir arrays

# Autoboxing

- Java converte tipos primitivos em tipos classe em tempo de compilacao

```
List<Integer> li = new ArrayList<>();  
for (int i = 1; i < 50; i += 2)  
    li.add(i);
```



```
List<Integer> li = new ArrayList<>();  
for (int i = 1; i < 50; i += 2)  
    li.add(Integer.valueOf(i));
```

# Limitacoes do uso do parametro de tipo

- Dentro da especificacao da classe parametrizada, existem locais onde uma definicao de classe ordinaria pode ser usada, mas um parametro de tipo nao pode ser usado;
- Em particular, o parametro de tipo nao pode ser usado em expressoes usando “new” para criar novos objetos
  - Por exemplo, um parametro de tipo nao pode ser usado como um nome de construtor, ou como um construtor:

```
T object = new T() ;  
T[] a = new T[10] ;
```

# Limitacoes da Instanciacao da Classe Generica

- Arrays definidos desta forma sao ilegais:

```
Pair<String>[] a =  
    new Pair<String>[10];
```

- Embora pareça razoavel ter este tipo de declaracao, a forma como Java implementou classes genericas nao permite isso

# Usando Classes Genericas e AutoBoxing

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.*



# Usando Classes Genericas e AutoBoxing

## Program Output:

Display 14.7 Using Our Ordered Pair Class and Automatic Boxing

---

### SAMPLE DIALOGUE

Enter two numbers:

42 24

You guessed the secret numbers  
in the correct order!

# Multiplos Parametros de Tipos

- Uma classe generica pode possuir um numero indefinido de parametros de tipo.

```
class name<T1, T2, ..., Tn> { /* ... */ }
```

- Os parametros de tipos sao listados entre o simbolo menor-maior separados por virgula.

# Multiplos Parametros de Tipo (cont)

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     }
```

(continued)

# Multiplos Parametros de Tipo (cont)

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.*

# Classes Genericas e Expressoes

- Não é permitido criar uma classe generica com `Exception`, `Error`, `Throwable`, ou qualquer descendente da classe `Throwable`
  - Uma classe generica não pode ser criada se seus objetos forem throwable

```
public class GEx<T> extends Exception
```

- O exemplo acima gerara uma mensagem de erro de compilacao.

# Usando uma classe generica com dois parametros de tipos

**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  }
```

**Saida do Programa:**

## **SAMPLE DIALOGUE**

```
Our current rating for The Car Guys
is 8
How would you rate them?
10
Our new rating for The Car Guys
is 10
```

# Parametro de Tipos Limitados

- Em alguns casos é necessario restringir os tipos possiveis para um dado parametro de tipo  $T$ 
  - Por exemplo, para garantir que somente classe que implementam a interface **Comparable** serao plugados no parametro de tipo  $T$ , definindo a classe da seguinte forma:

```
public class RClass<T implements Comparable>
```

- "implements Comparable" serve como limite para o parametro de tipo  $T$ .
- Qualquer tentativa e plugar um tipo  $T$ , o qual nao implemente a interface **Comparable** resultara em um erro de compilacao.

# Parametro de Tipos Limitados (Cont)

- Um limite para um tipo pode ser um nome de uma classe, ao inves de uma interface
  - Desta forma, somente descendentes da classe limitante poderao ser plugados nos parametros de tipo:

```
public class ExClass<T extends Class1>
```

- Uma expressao de limite pode conter varias interfaces e até uma classe.
- Se existe mais de um parametro de tipo, a sintaxe sera' :

```
public class Two<T1 extends Class1, T2 extends Class2 &  
    Comparable>
```



# Parametro de Tipos Limitados (Cont)

## Display 14.10 A Bounded Type Parameter

---

```
1  public class Pair<T extends Comparable>
2  {
3      private T first;
4      private T second;

5      public T max()
6      {
7          if (first.compareTo(second) <= 0)
8              return first;
9          else
10             return second;
11     }
```

<All the constructors and methods given in Display 14.5  
are also included as part of this generic class definition>

```
12 }
```

```
Class A { /* ... */ }
interface B { /* ... */ }
interface C { /* ... */ }
```

```
class D <T extends A & B & C> { /* ... */ }
```

MULTIPLOS  
LIMITES

# Interfaces Genericas

- Uma interface pode ter um ou mais parametros de tipo.
- Os detalhes e notacao sao os mesmos usados nas definicoes de classes genericas.

# Metodos Genericos

- Quando uma classe generica é definida, o tipo de parametro pode ser usado nas definicoes dos seus metodos.
- Além disso, um metodo generico pode definir parametros de tipos que nao sao parametros de tipo da classe generica
  - Um metodo generico pode ser membro de um classe ordinaria ou classe generica que possui parametros diferentes.
  - O parametro de tipo é local para o metodo, e nao é valido para classe inteira.

# Metodos Genericos (Cont)

- O parametro de tipo deve ser colocado depois dos modificadores, antes do tipo de retorno da assinatura do metodo:

```
public static <T> T genMethod(T[] a)
```

- Quando um desses metodos genericos for invocado, o nome do método deve prefixado com o tipo a ser plugado, encerrado entre simbolo menor-maior

```
String s = NonG.<String>genMethod(c) ;
```

# Heranca com classes genericas

- Uma classe generica pode ser definida como um classe derivada de uma classe ordinaria ou de outra classe generica
  - Como em uma classe ordinaria, um objeto de um tipo subclasse é tambem do tipo da superclasse
- Dadas duas classes: **A** e **B**, e dado **G**: uma classe generica, nao existe nenhum relacionamento entre **G<A>** e **G<B>**
  - Isto é verdadeiro independente das classes **A** e **B**, isto é, se classe **B** é subclasse da classe **A**

# A classe generica derivada: Um exemplo

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 }
```

# A classe generica derivada: Um exemplo (cont.)

## 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 }
```

**Saida do Programa:**

### SAMPLE DIALOGUE

peanuts and beer is the same as  
beer and peanuts

# Diamante

- No Java 7, é possível usar o simbolo diamante (<>) no construtor da classe generica, omitindo o tipo passado como parametro para classe generica, como no exemplo abaixo:

```
Box<Integer> integerBox = new Box<>();
```

- O compilador pode inferir o tipo da instancia a partir da declaracao do objeto.



# Tipos Raw (brutos)

- É possível declarar e instanciar classes genericas sem passar os tipos como parametros.
- Neste caso, as instancias dessas classes genericas sao chamadas de Raw Types do seu tipo generico.

```
public class Box<T> {  
    public void set(T t) { /* ... */ }  
    // ...  
}
```

```
Box rawBox = new Box(); // Raw Type  
// Box é um Raw Type da Classe Generica Box(T)
```