



GENÉRICOS (CAPÍTULO 18 DEITEL)

Carla Merkle Westphall
INE-CTC-UFSC
E-Mail: carlamw@inf.ufsc.br
URL: <http://moodle.ufsc.br>
INE5605-Turma 0238A

OBJETIVOS

Neste capítulo vamos aprender:

- Criar métodos genéricos que executam tarefas idênticas sobre argumentos de tipos diferentes.
- Criar uma classe genérica `Stack` que pode ser usada para armazenar objetos de qualquer tipo de classe ou interface.

Introdução

- **Genéricos**

- **Nova característica do J2SE 5.0**
- **Oferece segurança de tipo em tempo de compilação**
 - **Captura tipos inválidos em tempo de compilação**
- **Métodos genéricos**
 - **Uma declaração de um método cujo tipos dos argumentos é definido em tempo de compilação**
 - **Uma única declaração de método implica em um conjunto de métodos relacionados**
- **Classes genéricas**
 - **Uma única declaração de classe implica em um conjunto de classes relacionadas**

Motivação para métodos genéricos

- **Métodos sobrecarregados**
 - Executam operações similares sobre tipos de dados diferentes
 - Exemplo: métodos sobrecarregados `printArray`
 - Array de `Integer`
 - Array de `Double`
 - Array de `Character`
 - Apenas **tipos referência** podem ser usados com métodos genéricos e classes genéricas

Resumo

```
1 // Arquivo OverloadedMethods.java
2 // Usando métodos sobrecarregados para imprimir arrays de tipos diferentes.
3
4 public class OverloadedMethods
5 {
6     // método printArray para imprimir array de Integer
7     public static void printArray( Integer[] inputArray )
8     {
9         // imprime elementos do array
10        for ( Integer element : inputArray )
11            System.out.printf( "%s ", element );
12
13        System.out.println();
14    } // fim do método printArray
15
16    // método printArray para imprimir array de Double
17    public static void printArray( Double[] inputArray )
18    {
19        // imprime elementos do array
20        for ( Double element : inputArray )
21            System.out.printf( "%s ", element );
22
23        System.out.println();
24    } // imprime elementos do array
25
```

OverloadedMethods
.java

Método printArray aceita
um array de objetos Integer

Linha 7

Linha 17

Método printArray aceita
um array de objetos Double



Resumo

LoadedMethods

.java

(2 de 3)

Linha 27

```
26 // método printArray para imprimir array de Character
27 public static void printArray( Character[] inputArray )
28 {
29     // imprime elementos do array
30     for ( Character element : inputArray )
31         System.out.printf( "%s ", element );
32
33     System.out.println();
34 } // fim do método printArray
35
36 public static void main( String args[] )
37 {
38     // cria arrays de Integer, Double e Character
39     Integer[] integerArray = { 1, 2, 3, 4, 5, 6 };
40     Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
41     Character[] characterArray = { 'H', 'E', 'L', 'L', 'O' };
42
```

Método printArray aceita
um array de objetos
Character



```

43 System.out.println( "Array integerArray contains: " );
44 printArray( integerArray ); // passa um array de Integer
45 System.out.println( "\nArray doubleArray contains: " );
46 printArray( doubleArray ); // passa um array de Double
47 System.out.println( "\nArray characterArray contains: " );
48 printArray( characterArray ); // passa um array de Character
49 } // fim do main
50 } // fim da classe OverloadedMethods

```

```

Array integerArray contains:
1 2 3 4 5 6

```

```

Array doubleArray contains:
1.1 2.2 3.3 4.4 5.5 6.6 7.7

```

```

Array characterArray contains:
H E L L O

```

Resumo

Em tempo de compilação, o compilador determina o argumento do tipo `integerArray` (i.e., `Integer[]`), tenta localizar um método chamado `printArray` que especifica um único parâmetro `Integer[]` (linhas 7-14)

Em tempo de compilação, o compilador determina o argumento do tipo `doubleArray` (i.e., `Double[]`), tenta localizar um método chamado `printArray` que especifica um único parâmetro `Double[]` (linhas 17-24)

Em tempo de compilação, o compilador determina o argumento do tipo `characterArray` (i.e., `Character[]`), tenta localizar um método chamado `printArray` que especifica um único parâmetro `Character[]` (linhas 27-34)

hods

48

do programa



Métodos Genéricos: Implementação

- **Criando um método genérico `printArray`**
 - Chamadas são idênticas
 - Saídas são idênticas
- **Declaração do método genérico**
 - Seção parâmetros de tipo
 - Delimitado por “<” e “>”
 - Precede o tipo de retorno do método
 - Contém um ou mais parâmetros de tipo

Resumo

```
1 // Arquivo: GenericMethodTest.java
2 // Usando métodos genéricos para imprimir array de diferentes tipos.
3
4 public class GenericMethodTest
5 {
6     // método genérico printArray
7     public static < E > void printArray( E[] inputArray )
8     {
9         // apresenta elementos do array
10        for ( E element : inputArray )
11            System.out.printf( "%s ", element );
12
13        System.out.println();
14    } // fim do método printArray
15
16    public static void main( String args[] )
17    {
18        // cria arrays de Integer, Double e Character
19        Integer[] intArray = { 1, 2, 3, 4, 5 };
20        Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
21        Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };
22    }
```

Usa o parâmetro tipo para declarar o tipo de parâmetro do método

Seção parâmetro tipo delimitada por < e >

Usa o parâmetro tipo para declarar a variável local do método printArray

GenericMethodTest
.java

(1 de 2)

Linha 7

Linha 7

Linha 10



```

23 System.out.println( "Array integerArray contains: " );
24 printArray( integerArray ); // pass an Integer array
25 System.out.println( "\nArray doubleArray contains: " );
26 printArray( doubleArray ); // pass a Double array
27 System.out.println( "\nArray characterArray contains: " );
28 printArray( charArray ); // pass a Character array
29 } // fim do main
30 } // fim da classe GenericMethodTest

```

```

Array integerArray contains:
1 2 3 4 5 6

Array doubleArray contains:
1.1 2.2 3.3 4.4 5.5 6.6 7.7

Array characterArray contains:
H E L L O

```

Invoca método genérico
printArray com um array
Integer

Invoca método genérico printArray
com um array Double

Invoca método genérico printArray
com um array Character

Resumo

GenericMethodTest

Java

de 2)

Linha 24

Linha 26

Linha 28

Saida do programa



Boa prática de programação

É recomendado que os parâmetros de tipo sejam especificados como letras individuais em maiúscula. Tipicamente, um parâmetro de tipo que representa o tipo de um elemento em um array (ou outra coleção) é nomeado E para “elemento.”

Métodos genéricos: Implementação e tradução em tempo de compilação

- **Parâmetro de tipo:**
 - Também conhecido como variável de tipo.
 - Um identificador que especifica um nome de tipo genérico.
 - Utilizado para declarar o tipo de retorno, tipos de parâmetro e tipos de variáveis locais.
 - Funciona como marcadores de lugar para os tipos de argumento passados para o método genérico.
 - Argumentos de tipo reais.
 - Podem ser declarados somente uma vez, mas podem aparecer mais de uma vez, por exemplo:
 - `public static < E > void printTwoArrays(E[] array1, E[] array2)`.

Métodos genéricos: implementação e tradução em tempo de compilação

- **Tradução em tempo de compilação:**
 - **Erase:**
 - Remove a seção de parâmetro de tipo.
 - Substitui parâmetros de tipo por tipos reais.
 - Tipo padrão é Object.

```
1 public static void printArray( Object[] inputArray )
2 {
3     // exibe elementos do array
4     for ( Object element : inputArray )
5         System.out.printf( "%s\n", element );
6
7     System.out.println();
8 } // fim do método printArray
```

Remove a seção de parâmetro de tipo e substitui o parâmetro de tipo pelo `Object` do tipo real

Substitui o parâmetro de tipo pelo tipo `Object` real

O método genérico `printArray` depois de a erasure ser realizada pelo compilador.



Métodos que utilizam um parâmetro de tipo como o tipo de retorno

- **Aplicativo da Figura 18.5:**
 - Método genérico.
 - Utiliza os parâmetros de tipo no tipo de retorno e na lista de parâmetros.
- **Interface genérica:**
 - Especifica, com uma única declaração de interface, um conjunto de tipos relacionados.
 - Por exemplo, `Comparable< T >`.
 - Método `integer1.compareTo(integer2)`:
 - compara dois objetos da mesma classe;
 - retorna 0 se dois objetos forem iguais;
 - retorna -1 se `integer1` for menor que `integer2`; e
 - retorna 1 se `integer1` for maior que `integer2`.

```

1 // Fig. 18.5: MaximumTest.java
2 // O método genérico máximo retorna o maior dos três
3
4 public class MaximumTest
5 {
6     // determina o maior dos três objetos Comparable
7     public static < T extends Comparable< T > > T máximo( T x, T y, T z )
8     {
9         T max = x; // assume que x é inicialmente o maior
10
11         if ( y.compareTo( max ) > 0 )
12             max = y; // y é o maior até agora
13
14         if ( z.compareTo( max ) > 0 )
15             max = z; // z é o maior
16
17         return max; // retorna o maior objeto
18     } // fim do método máximo
19

```

A seção do parâmetro de tipo especifica que somente o objeto das classes que implementam a interface Comparable pode ser utilizado com esse método

MaximumTest.java

O parâmetro de tipo é utilizado no tipo de retorno do método máximo

Atribui x à variável local max

Invoca o método Comparable do método compareTo para comparar y e max

Invoca o método Comparable do método compareTo para comparar z e max

Linhas 14-15



Resumo

Maxi mumTest. j ava

```
20 public static void main( String args[] )
21 {
22     System.out.printf( "Maxi mum of %d, %d and %d is %d\n\n", 3, 4, 5,
23         maxi mum( 3, 4, 5 ) );
24     System.out.printf( "Maxi mum of %.1f, %.1f and %.1f is %.1f\n\n",
25         6.6, 8.8, 7.7, maxi mum( 6.6, 8.8, 7.7 ) );
26     System.out.printf( "Maxi mum of %s, %s and %s is %s\n\n",
27         "apple", "orange", maxi mum( "pear", "apple", "orange" ) );
28 } // fim do main
29 } // fim da classe Maxi mumTest
```

Invoca o método genérico
maxi mum com três inteiros

Invoca o método genérico
maxi mum com três doubles

Invoca o método genérico
maxi mum com três strings

Maxi mum of 3, 4 and 5 is 5

Maxi mum of 6.6, 8.8 and 7.7 is 8.8

Maxi mum of pear, apple and orange is pear

Linha 27

Saída do programa



Métodos que utilizam um parâmetro de tipo como o tipo de retorno (*Cont.*)

- **Limite superior do parâmetro de tipo:**
 - O padrão é `Object`.
 - Sempre utilize a palavra-chave `extends`. Por exemplo: `T extends Comparable< T >`.
 - Quando o compilador traduz um método genérico para bytecode Java:
 - substitui o parâmetro de tipo pelo seu limite superior; e
 - insere a operação de coerção explícita. Por exemplo, a linha 23 da Figura 18.5, é precedida por uma coerção `Integer` (`Integer`) `maximum(3, 4, 5)`.

Resumo

```
1 public static Comparable maximum(Comparable x, Comparable y, Comparable z)
2 {
3     Comparable max = x; // supõe que x é o maior
4
5     if ( y.compareTo( max ) > 0 )
6         max = y; // y é o maior
7
8     if ( z.compareTo( max ) > 0 )
9         max = z; // z é o maior
10
11     return max; // retorna o maior objeto
12 } // fim do método maximum
```

A erasure substitui o parâmetro do tipo T pelo seu limite superior Comparable

A erasure substitui o parâmetro do tipo T pelo seu limite superior Comparable

Classes genéricas

- **Classes genéricas**
 - Usa uma notação simples e concisa para indicar os tipos reais
- **Declaração de uma classe genérica**
 - Similar a uma declaração de classe não genérica
 - Exceto o nome da classe seguido de uma seção parâmetro de tipo

Classes genéricas

- **Classe genérica em tempo de compilação:**
 - O compilador realiza uma erasure nos parâmetros de tipo da classe.
 - O compilador substitui os parâmetros de tipo pelos seus limites superiores.
- **Programa de teste de classe genérica em tempo de compilação:**
 - O compilador realiza uma verificação de tipos.
 - O compilador insere as operações de coerção conforme necessário.

```

1 // Arquivo: Stack.java
2 // Classe genérica Stack.
3
4 public class Stack< E >
5 {
6     private final int size; // número de elementos
7     private int top; // Localização do elemento topo
8     private E[] elements; // array que armazena os elementos da pilha
9
10    // construtor sem argumento cria um array do tipo E
11    public Stack()
12    {
13        this( 10 ); // tamanho default da pilha
14    } // fim do construtor sem argumento
15
16    // construtor cria uma pilha com um número especificado de elementos
17    public Stack( int s )
18    {
19        size = s > 0 ? s : 10; // configura tamanho da pilha
20        top = -1; // pilha inicialmente vazia
21
22        elements = ( E[] ) new Object[ size ]; // cria array
23    } // fim do construtor
24

```

Declaração de classe genérica, nome da classe é seguida da seção parâmetro tipo

Declara elements como um array que armazena objetos do tipo E

Cria um array do tipo E. O mecanismo genérico não permite o uso do parâmetro tipo na criação da array pois o parâmetro tipo não é disponível em tempo de execução

Resumo

Stack.java

(1 de 2)

Linha 4

Linha 8

Linha 22



Resumo

Stack.java

(2 de 2)

Linhas 27-34

Linhas 37-43

```
25 // coloca elemento na pilha; se com sucesso, retorna true;
26 // senão, dispara FullStackException
27 public void push( E pushValue )
28 {
29     if ( top == size - 1 ) // se a pilha é cheia
30         throw new FullStackException( String.format(
31             "Stack is full, cannot push %s", pushValue ) );
32
33     elements[ ++top ] = pushValue; // coloca pushValue na pilha
34 } // fim do método push
35
36 // retorna o elemento topo se não vazia; senão dispara EmptyStackException
37 public E pop()
38 {
39     if ( top == -1 ) // se a pilha é vazia
40         throw new EmptyStackException( "Cannot pop" );
41
42     return elements[ top-- ]; // remove e retorna o elemento topo da pilha
43 } // fim do método pop
44 } // fim da classe Stack< E >
```

Método push coloca elemento do tipo E na pilha

Método pop retorna o elemento topo, que é do tipo E



```

1 // Arquivo: FullStackException.java
2 // Indica que a pilha esta cheia.
3 public class FullStackException extends RuntimeException
4 {
5     // construtor sem argumento
6     public FullStackException()
7     {
8         this( "Stack is full" );
9     } // fim do construtor sem argumento FullStackException
10
11     // construtor com um argumento
12     public FullStackException( String exception )
13     {
14         super( exception );
15     } // fim do construtor FullStackException
16 } // fim da classe FullStackException

```

Resumo

FullStack
Exception.java




```

1 // Arquivo: EmptyStackException.java
2 // Indica que a pilha esta cheia.
3 public class EmptyStackException extends RuntimeException
4 {
5     // construtor sem argumento
6     public EmptyStackException()
7     {
8         this( "Stack is empty" );
9     } // fim do construtor EmptyStackException
10
11     // construtor com um argumento
12     public EmptyStackException( String exception )
13     {
14         super( exception );
15     } // fim do construtor EmptyStackException
16 } // fim da classe EmptyStackException

```

Resumo

EmptyStack
Exception.java



```

1 // Arquivo: StackTest.java
2 // Programa teste da classe genérica Stack.
3
4 public class StackTest
5 {
6     private double[] doubleElements = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6 };
7     private int[] integerElements = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
8
9     private Stack< Double > doubleStack; // pilha de objetos Double
10    private Stack< Integer > integerStack; // pilha de objetos Integer
11
12    // testa objetos Stack
13    public void testStacks()
14    {
15        doubleStack = new Stack< Double >( 5 ); // Pilha de Double
16        integerStack = new Stack< Integer >( 10 ); // Pilha de Integer
17
18        testPushDouble(); // coloca double na doubleStack
19        testPopDouble(); // retira da doubleStack
20        testPushInteger(); // coloca int na intStack
21        testPopInteger(); // retira da intStack
22    } // fim do método testStacks
23

```

Classe genérica Stack com
argumento tipo Double

Classe genérica Stack com
argumento tipo Integer

Instancia objeto doubleStack de
tamanho 5 e integerStack de
tamanho 10

Resumo

StackTest.java

(1 de 6)

Linha 9

Linha 10

Linha 15-26



Resumo

StackTest.java

(2 de 6)

Linha 36

```
24 // testa método push com uma pilha double
25 public void testPushDouble()
26 {
27     // coloca elementos na pilha
28     try
29     {
30         System.out.println( "\nPushing elements onto doubleStack" );
31
32         // Coloca elementos na pilha
33         for ( double element : doubleElements )
34         {
35             System.out.printf( "%.1f ", element );
36             doubleStack.push( element ); // push onto doubleStack
37         } // fim do for
38     } // fim do try
39     catch ( FullStackException fullStackException )
40     {
41         System.err.println();
42         fullStackException.printStackTrace();
43     } // fim do catch FullStackException
44 } // fim do método testPushDouble
45
```

Invoca método push da Stack para colocar um valor double em doubleStack

Resumo

StackTest.java

(3 de 6)

Linha 58

```
46 // testa método pop com uma pilha double
47 public void testPopDouble()
48 {
49     // retira elementos da pilha
50     try
51     {
52         System.out.println( "\nPopping elements from doubleStack" );
53         double popValue; // armazena elemento removido da pilha
54
55         // remove todos os elementos da pilha
56         while ( true )
57         {
58             popValue = doubleStack.pop(); // retira da pilha doubleStack
59             System.out.printf( "%.1f ", popValue );
60         } // fim do while
61     } // fim do try
62     catch( EmptyStackException emptyStackException )
63     {
64         System.err.println();
65         emptyStackException.printStackTrace();
66     } // fim do catch EmptyStackException
67 } // fim do método testPopDouble
68
```

Retira valor Double da pilha e atribui a uma variavel local

Resumo

StackTest.java

(4 de 6)

Linha 81

```
69 // testa método push com uma pilha integer
70 public void testPushInteger()
71 {
72     // coloca elementos na pilha
73     try
74     {
75         System.out.println( "\nPushing elements onto intStack" );
76
77         // coloca elementos na pilha
78         for ( int element : integerElements )
79         {
80             System.out.printf( "%d ", element );
81             integerStack.push( element ); // coloca na integerStack
82         } // fim do for
83     } // fim do try
84     catch ( FullStackException fullStackException )
85     {
86         System.err.println();
87         fullStackException.printStackTrace();
88     } // fim do catch FullStackException
89 } // fim do método testPushInteger
90
```

Invoca método Stack push para colocar um valor int na integerStack

Resumo

StackTest.java

(5 de 6)

Linha 103

```
91 // testa método pop para retirar valor da pilha de Integer
92 public void testPopInteger()
93 {
94     // retira elementos da pilha
95     try
96     {
97         System.out.println( "\nPoppi ng el ements from i ntStack" );
98         int popVal ue; // armazena el emento removi do da pil ha
99
100        // remove todos os el ementos da pil ha
101        while ( true )
102        {
103            popVal ue = integerStack.pop(); // retira da i ntStack
104            System.out.pri nt f( "%d ", popVal ue );
105        } // fim do while
106    } // fim do try
107    catch( EmptyStackExcepti on emptyStackExcepti on )
108    {
109        System.err.pri nt l n();
110        emptyStackExcepti on.pri ntStackTrace();
111    } // fim do catch EmptyStackExcepti on
112 } // fim do método testPopInteger
113
114 public static void main( String args[] )
115 {
116     StackTest appli cation = new StackTest();
117     appli cation.testStacks();
118 } // fim do main
119 } // fim da classe StackTest
```

Retira um valor Integer da pilha



Pushing elements onto doubleStack

1.1 2.2 3.3 4.4 5.5 6.6

FullStackException: Stack is full, cannot push 6.6

```
at Stack.push(Stack.java:30)
at StackTest.testPushDouble(StackTest.java:36)
at StackTest.testStacks(StackTest.java:18)
at StackTest.main(StackTest.java:117)
```

Popping elements from doubleStack

5.5 4.4 3.3 2.2 1.1

EmptyStackException: Stack is empty, cannot pop

```
at Stack.pop(Stack.java:40)
at StackTest.testPopDouble(StackTest.java:58)
at StackTest.testStacks(StackTest.java:19)
at StackTest.main(StackTest.java:117)
```

Pushing elements onto integerStack

1 2 3 4 5 6 7 8 9 10 11

FullStackException: Stack is full, cannot push 11

```
at Stack.push(Stack.java:30)
at StackTest.testPushInteger(StackTest.java:81)
at StackTest.testStacks(StackTest.java:20)
at StackTest.main(StackTest.java:117)
```

Popping elements from integerStack

10 9 8 7 6 5 4 3 2 1

EmptyStackException: Stack is empty, cannot pop

```
at Stack.pop(Stack.java:40)
at StackTest.testPopInteger(StackTest.java:103)
at StackTest.testStacks(StackTest.java:21)
at StackTest.main(StackTest.java:117)
```

Resumo

StackTest.java

(6 de 6)

Saida do programa



Classes genéricas *(Exemplo: Pilha-Métodos Genéricos)*

- Criando métodos genéricos para testar a classe `Stack< E >`:
 - Método `testPush`:
 - Realiza as mesmas tarefas de `testPushDoub` e `testPushI nteger`.
 - Método `testPop`:
 - Realiza as mesmas tarefas de `testPopDoub` e `testPopI nteger`.