18

Genéricos

OBJETIVOS

- Neste capítulo, você aprenderá:
- Como criar métodos genéricos que realizam tarefas idênticas em argumentos de diferentes tipos.
- Como criar uma classe `Stack` genérica que pode ser utilizada para armazenar objetos de qualquer tipo de classe ou interface.
- Como sobrecarregar métodos genéricos com métodos não-genéricos ou com outros métodos genéricos.
- Como entender tipos brutos e como eles ajudam a alcançar a retrocompatibilidade.
- Como utilizar curingas quando informações precisas de tipo sobre um parâmetro não são requeridas no corpo do método.
- O relacionamento entre herança e genéricos.

Sumário

18.1 Introdução

18.2 Motivação para métodos genéricos

18.3 Métodos genéricos: implementação e tradução em tempo de compilação

18.4 Questões adicionais da tradução em tempo de compilação: métodos que utilizam parâmetro de tipo como o tipo de retorno

18.5 Sobrecarregando métodos genéricos

18.6 Classes genéricas

18.7 Tipos brutos

18.8 Curingas em métodos que aceitam parâmetros de tipo

18.9 Genéricos e herança: Notas

18.10 Conclusão

18.11 Internet e recursos da Web

18.1 Introdução

- **Genéricos:**
  - **Novo recurso do J2SE 5.0.**
  - **Fornece segurança de tipo em tempo de compilação:**
    - **Captura tipos inválidos em tempo de compilação.**
  - **Métodos genéricos:**
    - **Uma única declaração de método.**
    - **Um conjunto de métodos relacionados.**
  - **Classes genéricas:**
    - **Uma única declaração de classe.**
    - **Um conjunto de classes relacionadas.**

Observação de engenharia de software 18.1

Métodos e classes genéricas estão entre as capacidades mais poderosas do Java para reutilização de software com segurança de tipo em tempo de compilação.

18.2 Motivação para métodos genéricos

- **Métodos sobrecarregados:**
  - **Realiza operações semelhantes nos diferentes tipos de dados.**
  - **Métodos `printArray` sobrecarregados:**
    - **Array `Integer`.**
    - **Array `double`.**
    - **Array `character`.**
  - **Somente tipos por referência podem ser utilizados com métodos e classes genéricas.**

Resumo

```
1  // Fig. 18.1: OverloadedMethods.java
2  // Usando métodos sobrecarregados para imprimir um array de diferentes tipos.
3
4  public class OverloadedMethods
5  {
6     // método printArray para imprimir o array Integer
7     public static void printArray( Integer[] inputArray )
8     {
9        // exibe 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 o array Double
17    public static void printArray( Double[] inputArray )
18    {
19       // exibe elementos do array
20       for ( Double element : inputArray )
21          System.out.printf( "%s ", element );
22
23       System.out.println();
24    } // fim do método printArray
25
```

O método `printArray` aceita um array de objetos `Integer`

O método `printArray` aceita um array de objetos `Double`

`OverloadedMethods.java`

Linha 7

Linha 17

Resumo

```
26    // método printArray para imprimir um array Character
27    public static void printArray( Character[] inputArray )
28    {
29       // exibe 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 do 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
```

O método `printArray` aceita um array de objetos `Character`

`baded Methods .java`

(2 de 3)

Linha 27

```
43        System.out.println( "Array integerArray contains:" );
44        printArray( integerArray ); // passa um array Integer
45        System.out.println( "\nArray doubleArr
46        printArray( doubleArray ); // passa um
47        System.out.println( "\nArray character
48        printArray( characterArray ); // passa
49     } // fim de 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
```

Em tempo de compilação, o compilador determina o tipo do argumento `integerArray` (isto é, `Interger[]`), tenta localizar um método denominado `printArray` que especifica um único parâmetro `Integer[]` (linhas 7-14)

Em tempo de compilação, o compilador determina o tipo do argumento `doubleArrays` (isto é, `Double[]`), tenta localizar um método denominado `printArray` que especifica um único parâmetro `Double[]` (linhas 17-24)

Em tempo de compilação, o compilador determina o tipo do argumento `characterArray` (isto é, `Character[]`), tenta localizar um método denominado `printArray` que especifica um único parâmetro `Character[]` (linhas 7-14)

Resumo

thods

(3 de 3)

Saída do programa

18.2 Motivação para métodos genéricos (*Cont.*)

- **Estudar todos os métodos `printArray`:**
  - **O tipo de elemento no array aparece em dois locais:**
    - **cabeçalho do método; e**
    - **cabeçalho da instrução `for`.**
- **Combina três métodos `printArray` em um:**
  - **Substitui os tipos de elemento por um nome genérico `E`.**
  - **Declara um dos métodos `printArray`.**
    - **Exibe a representação de string dos elementos em qualquer array.**

Resumo

```
1 public static void printArray( E[] inputArray )
2 {
3    // exibe elementos do array
4    for ( E element : inputArray )
5       System.out.pri
6
7    System.out.printl
8 } // fim do método printArray
```

Substitui o tipo de elemento por um único tipo genérico E

Substitui o tipo de elemento por um único tipo genérico E

Figura 18.2 | Método `printArray` em que nomes de tipos reais são substituídos pelo nome genérico E por convenção.

18.3 Métodos genéricos: Implementação e tradução em tempo de compilação

- **Reimplementa a Figura 18.1 utilizando um método genérico:**
  - **Chamadas de métodos são idênticas.**
  - **Saídas são idênticas.**
- **Declaração de método genérico:**
  - **Seção de parâmetro de tipo:**
    - **delimitado por colchetes angulares (<D e >);**
    - **precede o tipo de retorno do método;**
    - **contém um ou mais parâmetros de tipo; e**
    - **também é chamado de parâmetros de tipo formal.**

18.3 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 ).`

Erro comum de programação 18.1

Ao declarar um método genérico, não conseguir colocar uma seção de parâmetro de tipo antes do tipo de retorno de um método é um erro de sintaxe — o compilador não entenderá o nome do parâmetro de tipo quando ele for encontrado no método.

Resumo

```
1  // Fig. 18.3: GenericMethodTest.java
2  // Utilizando métodos genéricos para imprimir array de tipos diferentes.
3
4  public class GenericMethodTest
5  {
6     // método genérico printArray
7     public static < E > void printArray( E[] inputArray )
8     {
9        // exibe elementos do array
10       for ( E element : inputArray
11          System.out.printf( "%s "
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
```

Utiliza o parâmetro de tipo para declarar o tipo de parâmetro do método `printArray`

Seção do parâmetro de tipo delimitada por colchetes angulares (< e >)

Utiliza o parâmetro de tipo para declarar o tipo de variável local do método `printArray`

`.java`

(1 de 2)

Linha 7

Linhas 10

```
23       System.out.println( "Array integerArray contains:" );
24       printArray( integerArray ); // passa um array Integer
25       System.out.println( "\nArray doubleArray contains:" );
26       printArray( doubleArray ); // passa um array Double
27       System.out.println( "\nArray characterArray contains:" );
28       printArray( characterArray ); // passa um array Character
29    } // fim de 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 o método genérico `printArray` com um array `Integer`

Invoca o método genérico `printArray` com um array `Double`

Invoca o método genérico `printArray` com um array `Character`

Resumo

`GenericMethodTest`

Linha 24

Linha 28

Saída do programa

Boa prática de programação 18.1

É recomendável que os parâmetros de tipo sejam especificados como letras maiúsculas individuais. Em geral, um parâmetro de tipo que representa o tipo de um elemento em um array (ou outra coleção) é nomeado E, de ‘elemento’.

Erro comum de programação 18.2

Se o compilador não puder encontrar uma correspondência entre uma chamada de método e uma declaração de método genérico ou não-genérico, ocorrerá um erro de compilação.

Erro comum de programação 18.3

Se o compilador não encontrar uma declaração de método que corresponda exatamente a uma chamada de método, mas encontrar dois ou mais métodos genéricos que podem satisfazer a chamada de método, ocorrerá um erro de compilação.

18.3 Métodos genéricos: implementação e tradução em tempo de compilação (*Cont.*)

- **Tradução em tempo de compilação:**
  - **Erasure:**
    - **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 ", 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

Figura 18.4 | O método genérico `printArray` depois de a erasure ser realizada pelo compilador.

18.4 Questões adicionais da tradução em tempo de compilação: Métodos que utilizam um parâmetro de tipo como o tipo de retorno

- **Aplicativo da Figura 18.5:**
  - **Método genérico.**
  - **Utiliza os parâmetros Type 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 maximum retorna o maior dos trê
3
4  public class MaximumTest
5  {
6     // determina o maior dos três objetos Comparable
7     public static < T extends Comparable< T > > T maximum( 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 maximum
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 `maximum`

Linha 7

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

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

Invoca o método genérico `maximum` com três inteiros

Invoca o método genérico `maximum` com três doubles

Invoca o método genérico `maximum` com três strings

MaximumTest.java

Linha 25

Linha 27

Saída do programa

```
Maximum of 3, 4 and 5 is 5

Maximum of 6.6, 8.8 and 7.7 is 8.8

Maximum of pear, apple and orange is pear
```

18.4 Questões adicionais da tradução em tempo de compilação: 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, que 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
4
5     if ( y.compareTo( max ) > 0 )
6        max = y; // y é o maio
7
8     if ( z.compareTo( max ) >
9        max = z; // z é o maio
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

18.5 Sobrecarregando métodos genéricos

- **Método genérico pode ser sobrecarregado:**
  - **Por outro método genérico:**
    - **Mesmo nome de método, porém diferentes parâmetros de método.**
  - **Por métodos não-genéricos:**
    - **Mesmo nome de método e número de parâmetros.**
- **Quando o compilador encontra uma chamada de método:**
  - **Primeiro, procura o método mais preciso de correspondência:**
    - **Nome de método e tipos de argumentos exatos.**
  - **Então, procura o método inexato, mas aplicável de correspondência.**

18.6 Classes genéricas

- **Classes genéricas:**
  - **Utiliza uma notação simples, concisa, para indicar o(s) tipo(s) real(is).**
  - **Em tempo de compilação, o compilador Java:**
    - **assegura a segurança de tipos; e**
    - **utiliza a técnica de erasure para permitir que o código de cliente possa interagir com a classe genérica.**
- **Classes parametrizadas:**
  - **Também chamadas *tipos parametrizados*.**
  - **Por exemplo: `Stack< Double >`.**

18.6 Classes genéricas (*Cont.*)

- **Declaração de classe genérica:**
  - **É parecida a uma declaração de classe não-genérica.**
  - **Exceto que o nome da classe é seguido por uma seção de parâmetro de tipo.**
- **A opção `–Xlint:unchecked`:**
  - **O compilador não pode assegurar 100% de segurança com relação a tipos.**

Resumo

Stack.java

(1 de 2)

Linha 4

Linha 8

Linha 22

```
1  // Fig. 18.7: Stack.java
2  // Classe genérica Stack.
3
4  public class Stack< E >
5  {
6     private final int size; // número d
7     private int top; // localização do
8     private E[] elements; // array que
9
10    // construtor sem argumentos cria u
11    public Stack()
12    {
13       this( 10 ); // pilha do tamanho padrão
14    } // fim do construtor sem argumentos da classe Stack
15
16    // construtor cria uma pilha do número especificado de elementos
17    public Stack( int s )
18    {
19       size = s > 0 ? s : 10; // configura o tamanho de Stack
20       top = -1; // Stack inicialmente vazia
21
22       elements = ( E[] ) new Object[ size ]; // cria o array
23    } // fim do construtor Stack
24
```

Declaração de classe genérica, o nome de classe é seguido por uma seção de parâmetro de tipo

Declara elementos como um array que armazena objetos do tipo E

Cria um array do tipo E. O mecanismo genérico não permite parâmetro de tipo nas expressões de criação de array porque o parâmetro de tipo não está disponível em tempo de execução

```
25    // insere elemento na pilha; se bem-sucedido, retorna true;
26    // caso contrário, lança uma FullStackException
27    public void push( E pushValue )
28    {
29       if ( top == size - 1 ) // se a pilha es
30          throw new FullStackException( String
31             "Stack is full, cannot push %s", pushValue ) );
32
33       elements[ ++top ] = pushValue; // insere pushValue na Stack
34    } // fim do método push
35
36    // retorna o elemento superior se não estiver vazia; do contrário lança uma EmptyStackException
37    public E pop()
38    {
39       if ( top == -1 ) // se
40          throw new EmptyStac                    annot pop" );
41
42       return elements[ top-- ]; // remove e retorna elemento superior de Stack
43    } // fim do método pop
44 } // fim da classe Stack< E >
```

O método push coloca o elemento do tipo E na pilha

O método pop retorna o elemento superior, que é do tipo E

Resumo

Stack.java

(2 de 2)

Linhas 27-34

Linhas 37-43

Resumo

FullStack Exception.java

```
1  // Fig. 18.8: FullStackException.java
2  // Indica que a pilha está cheia.
3  public class FullStackException extends RuntimeException
4  {
5     // construtor sem argumentos
6     public FullStackException()
7     {
8        this( "Stack is full" );
9     } // fim do construtor FullStackException sem argumentos
10
11    // construtor de um argumento
12    public FullStackException( String exception )
13    {
14       super( exception );
15    } // fim do FullStackException de um argumento
16 } // fim da classe FullStackException
```

Resumo

EmptyStack
Exception.java

```
1  // Fig. 18.9: EmptyStackException.java
2  // Indica que uma pilha está cheia.
3  public class EmptyStackException extends RuntimeException
4  {
5     // construtor sem argumentos
6     public EmptyStackException()
7     {
8        this( "Stack is empty" );
9     } // fim do construtor EmptyStackException sem argumentos
10
11    // construtor sem argumentos
12    public EmptyStackException( String exception )
13    {
14       super( exception );
15    } // fim do construtor EmptyStackException de um argumento
16 } // fim da classe EmptyStackException
```

18.6 Classes genéricas (*Cont.*)

- **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.**

Resumo

Stacktest.java

(1 de 6)

Linha 9

Linha 10

Linhas 15-26

```
1  // Fig. 18.10: StackTest.java
2  // Programa de 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 armazena objetos Double
10    private Stack< Integer > integerS                    s Integer
11
12    // testa objetos Stack
13    public void testStacks()
14    {
15       doubleStack = new Stack< Double >( 5 );
16       integerStack = new Stack< Integer >( 10
17
18       testPushDouble(); // insere doubles em doubleStack
19       testPopDouble(); // remove de doubleStack
20       testPushInteger(); // insere ints em intStack
21       testPopInteger(); // remove de intStack
22    } // fim do método testStacks
23
```

O argumento de tipo da classe genérica `Stack` é `Double`

O argumento de tipo da classe genérica `Stack` é `Integer`

Instancia o objeto `doubleStack` de tamanho 5 e `integerStack` de tamanho 10

Resumo

Stacktest.java

(2 de 6)

Linha 36

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

Invoca o método `push` de `Stack` para colocar um valor `double` em `doubleStack`

Resumo

Stacktest.java

(3 de 6)

Linha 58

```
46    // testa o método pop com a pilha de doubles
47    public void testPopDouble()
48    {
49       // pop elements from stack
50       try
51       {
52          System.out.println( "\nPopping elements from doubleStack" );
53          double popValue; // armazena o elemento removido da pilha
54
55          // remove all elements from Stack
56          while ( true )
57          {
58             popValue = doubleStack.pop(); // remove de 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 da captura de EmptyStackException
67    } // fim do método testPopDouble
68
```

O auto-unboxing ocorre quando o valor retornado por `pop` (`Double`) é atribuído a uma variável primitiva `double`

Resumo

Stacktest.java

(4 de 6)

Linha 81

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

Invoca o método `push` de `Stack` a fim de colocar um valor `int` em `integerStack`

Resumo

Stacktest.java

(5 de 6)

Linha 103

```
91    // testa o método pop com a pilha de integers
92    public void testPopInteger()
93    {
94       // remove elementos da pilha
95       try
96       {
97          System.out.println( "\nPopping elements from intStack" );
98          int popValue; // armazena o elemento removido da pilha
99
100         // remove todos os elementos da Stack
101         while ( true )
102         {
103            popValue = integerStack.pop(); // remove de intStack
104            System.out.printf( "%d ", popValue );
105         } // fim do while
106      } // fim do try
107      catch( EmptyStackException emptyStackException )
108      {
109         System.err.println();
110         emptyStackException.printStackTrace();
111      } // fim da captura de EmptyStackException
112   } // fim do método testPopInteger
113
114   public static void main( String args[] )
115   {
116      StackTest application = new StackTest();
117      application.testStacks();
118   } // fim do main
119} // fim da classe StackTest
```

O auto-unboxing ocorre quando o valor retornado por `pop` (`Integer`) é atribuído a uma variável primitiva `int`

```
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)

Saída do programa

18.6 Classes genéricas (*Cont.*)

- **Criando métodos genéricos para testar a classe `Stack< E >`:**
  - **Método `testPush`:**
    - **Realiza as mesmas tarefas de `testPushDouble` e `testPushInteger`.**
  - **Método `testPop`:**
    - **Realiza as mesmas tarefas de `testPopDouble` e `testPopInteger`.**

Resumo

StackTest2.java

(1 de 4)

Linhas 19-22

```
1  // Fig. 18.11: StackTest2.java
2  // Programa de teste da classe genérica Stack.
3
4  public class StackTest2
5  {
6     private Double[] doubleElements = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6 };
7     private Integer[] integerElements =
8        { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
9
10    private Stack< Double > doubleStack; // a pilha armazena objetos Double
11    private Stack< Integer > integerStack; // a pilha armazena objetos Integer
12
13    // testa objetos Stack
14    public void testStacks()
15    {
16       doubleStack = new Stack< Double >( 5 ); // Stack de Doubles
17       integerStack = new Stack< Integer >( 10 ); // Stack de Integers
18
19       testPush( "doubleStack", doubleStack, doubleElements );
20       testPop( "doubleStack", doubleStack );
21       testPush( "integerStack", integerStack, integerEleme
22       testPop( "integerStack", integerStack );
23    } // fim do método testStacks
24
```

Invoca os métodos genéricos `testPush` e `testPop` para colocar elementos na pilha e remover elementos da pilha

Resumo

```
25    // método genérico testPush insere elementos em uma Stack
26    public < T > void testPush( String name, Stack< T > stack,
27       T[] elements )
28    {
29       // insere elementos na pilha
30       try
31       {
32          System.out.printf( "\nPushing elements onto %s\n", name );
33
34          // insere elementos na Stack
35          for ( T element : elements )
36          {
37             System.out.printf( "%s ", element );
38             stack.push( element ); // insere element
39          }
40       } // fim do try
41       catch ( FullStackException fullStackException )
42       {
43          System.out.println();
44          fullStackException.printStackTrace();
45       } // fim da captura de FullStackException
46    } // fim do método testPush
47
```

O método genérico `TestPush` substitui `testPushDouble` e `testPushInteger`

StackTest2.java

(2 de 4)

Linhas 26-27

Substitui o tipo de elemento `Double`/`Integer` pelo parâmetro de tipo `T`

```
48    // método genérico testPop remove elementos de uma Stack
49    public < T > void testPop( String name, Stack< T > stack )
50    {
51       // remove elementos da pilha
52       try
53       {
54          System.out.printf( "\nPopping elements from %s\n", name );
55          T popValue; // armazena o elemento removido da pilha
56
57          // remove elementos da Stac
58          while ( true )
59          {
60             popValue = stack.pop(); // remove da pilha
61             System.out.printf( "%s ", popValue );
62          } // fim do while
63       } // fim do try
64       catch( EmptyStackException emptyStackException )
65       {
66          System.out.println();
67          emptyStackException.printStackTrace();
68       } // fim da captura de EmptyStackException
69    } // fim do método testPop
70
71    public static void main( String args[] )
72    {
73       StackTest2 application = new StackTest2();
74       application.testStacks();
75    } // fim de main
76 } // fim da classe StackTest2
```

Resumo

O método genérico `TestPop` substitui `testPopDouble` e `testPopInteger`

StackTest2.java

(3 de 4)

Substitui o tipo de elemento `Double`/`Integer` pelo parâmetro de tipo `T`

inhas 49-50

Linha 55

```
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 StackTest2.testPush(StackTest2.java:38)
        at StackTest2.testStacks(StackTest2.java:19)
        at StackTest2.main(StackTest2.java:74)

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 StackTest2.testPop(StackTest2.java:60)
        at StackTest2.testStacks(StackTest2.java:20)
        at StackTest2.main(StackTest2.java:74)

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 StackTest2.testPush(StackTest2.java:38)
        at StackTest2.testStacks(StackTest2.java:21)
        at StackTest2.main(StackTest2.java:74)

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 StackTest2.testPop(StackTest2.java:60)
        at StackTest2.testStacks(StackTest2.java:22)
        at StackTest2.main(StackTest2.java:74)
```

Resumo

StackTest2.java

(4 de 4)

Saída do programa

18.7 Tipos brutos

- **Tipo bruto:**
  - **Permite instanciar a classe genérica sem especificar um argumento de tipo. Por exemplo:**
  - `Stack objectStack = new Stack( e );`
  - **Dizemos que `objectStack` tem um tipo bruto.**
  - **Importante para retrocompatibilidade com versões anteriores.**
  - **Uma variável `Stack` do tipo bruto pode ser atribuída a uma `Stack` que especifica um argumento de tipo.**
  - **Uma variável `Stack` que especifica um argumento de tipo que pode ser atribuído a uma `Stack` do tipo bruto.**
    - **Permitido, mas inseguro.**
    - **Utilize a opção –the `Xlint:unchecked` para compilar.**

Resumo

RawTypeTest.java

(1 de 5)

Linha 14

Linha 17

Linha 20

```
1  // Fig. 18.12: RawTypeTest.java
2  // Programa de teste de tipos brutos.
3
4  public class RawTypeTest
5  {
6     private Double[] doubleElements = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6 };
7     private Integer[] integerElements =
8        { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
9
10    // método para testar classes Stack com tipos brutos
11    public void testStacks()
12    {
13       // Pilha de tipos brutos atribuídos à classe Stack da variável de tipos brutos
14       Stack rawTypeStack1 = new Stack( 5 );
15
16       // Stack< Double > atribuído a Stack da variáve
17       Stack rawTypeStack2 = new Stack< Double >( 5 );
18
19       // Pilha de tipos crus atribuídos à variável Stack< Integer
20       Stack< Integer > integerStack = new Stack( 10 );
21
22       testPush( "rawTypeStack1", rawTypeStack1, doubleElements );
23       testPop( "rawTypeStack1", rawTypeStack1 );
24       testPush( "rawTypeStack2", rawTypeStack2, doubleElements );
25       testPop( "rawTypeStack2", rawTypeStack2 );
26       testPush( "integerStack", integerStack, integerElements );
27       testPop( "integerStack", integerStack );
28    } // fim do método testStacks
29
```

Instancia a classe genérica `Stack` com o tipo bruto

Atribui uma `Stack< Double >` à variável `rawTypeStack2`

Atribui uma `Stack` do tipo bruto a `Stack< Integer >`. Válido, porém perigoso

Resumo

RawTypeTest.java

(2 de 5)

```
30    // método genérico insere elementos na pilha
31    public < T > void testPush( String name, Stack< T > stack,
32       T[] elements )
33    {
34       // insere elementos na pilha
35       try
36       {
37          System.out.printf( "\nPushing elements onto %s\n", name );
38
39          // insere elementos na Stack
40          for ( T element : elements )
41          {
42             System.out.printf( "%s ", element );
43             stack.push( element ); // insere o elemento na pilha
44          } // fim do for
45       } // fim do try
46       catch ( FullStackException fullStackException )
47       {
48          System.out.println();
49          fullStackException.printStackTrace();
50       } // fim da captura de FullStackException
51    } // fim do método testPush
52
```

Resumo

RawTypeTest.java

(3 de 5)

```
53    // método genérico testPop remove elementos da pilha
54    public < T > void testPop( String name, Stack< T > stack )
55    {
56       // remove elementos da pilha
57       try
58       {
59          System.out.printf( "\nPopping elements from %s\n", name );
60          T popValue; // armazena o elemento removido da pilha
61
62          // remove elementos da Stack
63          while ( true )
64          {
65             popValue = stack.pop(); // remove da pilha
66             System.out.printf( "%s ", popValue );
67          } // fim do while
68       } // fim do try
69       catch( EmptyStackException emptyStackException )
70       {
71          System.out.println();
72          emptyStackException.printStackTrace();
73       } // fim da captura de EmptyStackException
74    } // fim do método testPop
75
76    public static void main( String args[] )
77    {
78       RawTypeTest application = new RawTypeTest();
79       application.testStacks();
80    } // fim do main
81 } // fim da classe RawTypeTest
```

```
Pushing elements onto rawTypeStack1
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 RawTypeTest.testPush(RawTypeTest.java:43)
        at RawTypeTest.testStacks(RawTypeTest.java:22)
        at RawTypeTest.main(RawTypeTest.java:79)

Popping elements from rawTypeStack1
5.5 4.4 3.3 2.2 1.1
EmptyStackException: Stack is empty, cannot pop
        at Stack.pop(Stack.java:40)
        at RawTypeTest.testPop(RawTypeTest.java:65)
        at RawTypeTest.testStacks(RawTypeTest.java:23)
        at RawTypeTest.main(RawTypeTest.java:79)

Pushing elements onto rawTypeStack2
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 RawTypeTest.testPush(RawTypeTest.java:43)
        at RawTypeTest.testStacks(RawTypeTest.java:24)
        at RawTypeTest.main(RawTypeTest.java:79)
```

Resumo

RawTypeTest.java

(4 de 5)

Saída do programa

```
Popping elements from rawTypeStack2
5.5 4.4 3.3 2.2 1.1
EmptyStackException: Stack is empty, cannot pop
        at Stack.pop(Stack.java:40)
        at RawTypeTest.testPop(RawTypeTest.java:65)
        at RawTypeTest.testStacks(RawTypeTest.java:25)
        at RawTypeTest.main(RawTypeTest.java:79)

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 RawTypeTest.testPush(RawTypeTest.java:43)
        at RawTypeTest.testStacks(RawTypeTest.java:26)
        at RawTypeTest.main(RawTypeTest.java:79)

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 RawTypeTest.testPop(RawTypeTest.java:65)
        at RawTypeTest.testStacks(RawTypeTest.java:27)
        at RawTypeTest.main(RawTypeTest.java:79)
```

Resumo

RawTypeTest.java

(5 de 5)

Saída do programa

Resumo

```
RawTypeTest.java:20: warning: unchecked assignment
found   : Stack
required: Stack<java.lang.Integer>
      Stack< Integer > integerStack = new Stack( 10 );
                                      ^
RawTypeTest.java:22: warning: [unchecked] unchecked method invocation:
<T>testPush(java.lang.String,Stack<T>,T[]) in RawTypeTest is applied to
(java.lang.String,Stack,java.lang.Double[])
      testPush( "rawTypeStack1", rawTypeStack1, doubleElements );
      ^
RawTypeTest.java:23: warning: [unchecked] unchecked method invocation:
<T>testPop(java.lang.String,Stack<T>) in RawTypeTest is applied to
(java.lang.String,Stack)
      testPop( "rawTypeStack1", rawTypeStack1 );
      ^
RawTypeTest.java:24: warning: [unchecked] unchecked method invocation:
<T>testPush(java.lang.String,Stack<T>,T[]) in RawTypeTest is applied to
(java.lang.String,Stack,java.lang.Double[])
      testPush( "rawTypeStack2", rawTypeStack2, doubleElements );
      ^
RawTypeTest.java:25: warning: [unchecked] unchecked method invocation:
<T>testPop(java.lang.String,Stack<T>) in RawTypeTest is applied to
(java.lang.String,Stack)
      testPop( "rawTypeStack2", rawTypeStack2 );
      ^
5 warnings
```

Figura 18.13 | Mensagem de alerta do compilador.

18.8 Curingas em métodos que aceitam parâmetros de tipo

- **Estrutura de dados `ArrayList`:**
  - **Estrutura de dados dinamicamente redimensionável semelhante a um array.**
  - **Método `add`.**
  - **Método `toString`.**
- **Motivação para o uso de curingas:**
  - **Implementa um método genérico `sum`.**
    - **Soma os números em uma coleção.**
    - **Recebe um parâmetro do tipo `ArrayList<Number>`.**
    - **Utiliza o método `doubleValue` da classe `Number` para obter o valor primitivo subjacente de `Numbers` como um valor `double`.**

Resumo

TotalNumbers.java

(1 de 2)

```
1  // Fig. 18.14: TotalNumbers.java
2  // Somando os elementos de um ArrayList.
3  import java.util.ArrayList;
4
5  public class TotalNumbers
6  {
7     public static void main( String args[] )
8     {
9        // cria, inicializa e gera saída de ArrayList de
10       // Integers e Doubles e então exibe o total dos e
11       Number[] numbers = { 1, 2.4, 3, 4.1 }; // Integers e Doubles
12       ArrayList< Number > numberList = new ArrayList< Number >();
13
14       for ( Number element : numbers )
15          numberList.add( element ); // insere cada núme
16
17       System.out.printf( "numberList contains:
18       System.out.printf( "Total of the elements
19          sum( numberList ) );
20    } // fim do main
21
```

Linha 11

Linha 12

Linha 19

Declara e inicializa o array de `numbers`

Declara e inicializa `numberList`, que armazena objetos `Number`

Adiciona elementos no array `numbers` a `ArrayList` `numberList`

Invoca o método `sum` para calcular o total dos elementos armazenados em `numberList`

Resumo

```
22    // calcula total de elementos de ArrayList
23    public static double sum( ArrayList< Number > list )
24    {
25       double total = 0; // inicializa total
26
27       // calcula a soma
28       for ( Number element : list )
29          total += element.doubleValue();
30
31       return total;
32    } // fim do método sum
33 } // fim da classe TotalNumbers
```

O método `sum` aceita um `ArrayList` que armazena objetos `Number`

(2 de 2)

Utiliza o método `doubleValue` da classe `Number` para obter o valor primitivo subjacente de `Numbers` como um valor `double`

Saída do programa

```
numberList contains: [1, 2.4, 3, 4.1]
Total of the elements in numberList: 10.5
```

18.8 Curingas em métodos que aceitam parâmetros de tipo (*Cont.*)

- **Implementando o método `sum` com um argumento do tipo curinga no seu parâmetro:**
  - **`Number` é a superclasse `Integer`.**
  - **`ArrayList<Number>` não é um supertipo de `ArrayList<Integer>`.**
  - **Não pode passar `ArrayList<Integer>` para o método `sum`.**
  - **Utiliza um curinga para criar uma versão mais flexível de `sum`.**
    - `ArrayList< ? extends Number >`.
    - **`?` Representa um 'tipo desconhecido'.**
    - **Argumento de tipo desconhecido deve ser `Number` ou uma subclasse de `Number`.**
    - **Não pode utilizar um curinga como um nome de tipo no corpo de um método.**

Resumo

WildcardTest.java

(1 de 3)

Linha 12

Linha 20

```
1  // Fig. 18.15: WildcardTest.java
2  // Programa de teste de curinga.
3  import java.util.ArrayList;
4
5  public class WildcardTest
6  {
7     public static void main( String args[] )
8     {
9        // cria, inicializa e gera saída de ArrayList de Integers, então
10       // exibe total dos elementOs
11       Integer[] integers = { 1, 2, 3, 4, 5 };
12       ArrayList< Integer > integerList = new ArrayList< Integer >();
13
14       // insere elementos na integerList
15       for ( Integer element : integers )
16          integerList.add( element );
17
18       System.out.printf( "integerList contains: %s\n", integerList );
19       System.out.printf( "Total of the elements in integerList: %.0f\n\n",
20          sum( integerList ) );
21
22       // cria, inicializa e gera saída do A
23       // exibe o total dos elementos
24       Double[] doubles = { 1.1, 3.3, 5.5 };
25       ArrayList< Double > doubleList = new ArrayList< Double >();
26
27       // insere os elementos na doubleList
28       for ( Double element : doubles )
29          doubleList.add( element );
30
```

Declara e cria `ArrayList` `integerList` para conter `Integer`s

Invoca o método `sum` para calcular o total dos elementos armazenados em `integerList`

Declara e cria `ArrayList` `doubleList` para conter `Double`s

Resumo

```
31      System.out.printf( "doubleList contains: %s\n", doubleList );
32      System.out.printf( "Total of the elements in doubleList: %.1f\n\n",
33         sum( doubleList ) );
34
35      // cria, inicializa e gera saída de
36      // Integers e Doubles e, então, exi
37      Number[] numbers = { 1, 2.4, 3, 4.1
38      ArrayList< Number > numberList = new ArrayList< Number >();
39
40      // insere elementos na numberList
41      for ( Number element : numbers )
42         numberList.add( element );
43
44      System.out.printf( "numberList contains: %s\n", numberList );
45      System.out.printf( "Total of the elements in numberList: %.1f\n",
46         sum( numberList ) );
47   } // fim de main
48
49   // calcula o total de elementos da pil
50   public static double sum( ArrayList< ?
51   {
52      double total = 0; // inicializa o total
53
```

ldcardTest.java

(2 de 3)

Linha 38

Linha 46

Linha 50

Invoca o método `sum` para calcular o total dos elementos armazenados em `doubleList`

Declara e cria `ArrayList integerList` para conter `Numbers`

Invoca o método `sum` para calcular o total dos elementos armazenados em `numberList`

Os tipos de elemento do argumento `ArrayList` não são conhecidos diretamente pelo método; eles são conhecidos como sendo pelo menos do tipo `Number`

```
54       // calcula a soma
55       for ( Number element : list )
56          total += element.doubleValue();
57
58       return total;
59    } // fim do método sum
60 } // fim da classe WildcardTest
```

```
integerList contains: [1, 2, 3, 4, 5]
Total of the elements in integerList: 15

doubleList contains: [1.1, 3.3, 5.5]
Total of the elements in doubleList: 9.9

numberList contains: [1, 2.4, 3, 4.1]
Total of the elements in numberList: 10.5
```

Resumo

WildcardTest.java

(3 de 3)

Saída do programa

Erro comum de programação 18.4

Utilizar um curinga na seção de parâmetro de tipo de um método ou utilizar um curinga como um tipo explícito de uma variável no corpo do método é um erro de sintaxe.

18.9 Genéricos e herança: Notas

- **Herança nos genéricos:**
  - **Uma classe genérica pode ser derivada de uma classe não-genérica.**
  - **Por exemplo: a classe `Object` é uma superclasse de todas as classes genéricas.**
  - **Uma classe genérica pode ser derivada de uma outra classe genérica.**
  - **Por exemplo: `Stack` é uma subclasse de `Vector`.**
  - **Uma classe não-genérica pode ser derivada de uma outra classe genérica.**
  - **Por exemplo: `Properties` é uma subclasse de `Hashtable`.**
  - **Um método genérico em uma subclasse pode sobrescrever um método genérico em uma superclasse.**
    - **Se os dois métodos tiverem a mesma assinatura.**