```
public class Box<T> {
    // T stands for "Type".
    private T t;
    public void set(T t) { this.t = t; }
    public T get( ) { return t; }
}
```

Aula 22

Classes e Métodos Genéricos

Rafael Geraldeli Rossi

Introdução

- Seria conveniente escrever um único método `sort` para classificar os elementos em um `array` de `Integer`, `String` ou de qualquer tipo que suporte ordenamento (possam ser comparáveis)
- Também seria conveniente escrever uma única classe `Stack` de qualquer tipo
- Igualmente útil seria detectar correspondências de tipos em tempo de compilação – **segurança de tipo em tempo de compilação** → por exemplo, se uma `Stack` armazenasse somente inteiros, ao tentar inserir uma `String` nessa `Stack` deveria emitir um erro em tempo de compilação

Introdução

- **Métodos genéricos** e **classes genéricas** (e interfaces) permitem especificar
  1. **Com uma única declaração de método um conjunto de métodos relacionados**
  2. **Com uma única declaração de classe um conjunto de tipos relacionados**

- Os genéricos também fornecem segurança de tipo em tempo de compilação que permite capturar tipos inválidos em tempo de compilação

Introdução

- Métodos sobrecarregados são frequentemente utilizados para realizar operações semelhantes em tipos diferentes de dados

```
11  public class MetodosSobrecarregados {
12
13      public static void main (String[] args){
14
15          Integer[] arrayInt = {1, 2, 3, 4, 5, 6};
16          Double[] arrayDbl = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7};
17          Character[] arrayCh = {'a', 'b', 'c', 'd', 'e', 'f'};
18
19          System.out.println("Imprimindo array de inteiros:");
20          printArray(arrayInt);
21          System.out.println("Imprimindo array de double:");
22          printArray(arrayDbl);
23          System.out.println("Imprimindo array de caracteres:");
24          printArray(arrayCh);
25      }
26
27      public static void printArray(Integer[] array){
29          for(int i=0;i<array.length;i++){
30              System.out.print(array[i] + "\t");
31          }
32          System.out.println();
33      }
34
36      public static void printArray(Double[] array){
37          for(int i=0;i<array.length;i++){
38              System.out.print(array[i] + "\t");
39          }
40          System.out.println();
41      }
42
43      public static void printArray(Character[] array){
44          for(int i=0;i<array.length;i++){
45              System.out.print(array[i] + "\t");
46          }
47          System.out.println();
48      }
49  }
```

MetodosSobrecarregados > printArray >

Resultados da Pesquisa | Saída - Teste (run) ×

```
run:
Imprimindo array de inteiros:
1       2       3       4       5       6
Imprimindo array de double:
1.1     2.2     3.3     4.4     5.5     6.6     7.7
Imprimindo array de caracteres:
a       b       c       d       e       f
```

Introdução

- Com a utilização de métodos sobrecarregados, podemos chamar um único método que irá dar conta de imprimir os três tipos diferentes de `arrays`
- Porém, tivemos que implementar os métodos `print` para os três tipos de `array`
- Se tivéssemos um quarto tipo, teríamos que implementar mais um método `print`
- Além disso, observem que os métodos são idênticos, apenas o tipo do parâmetro é diferente
- **Se substituíssemos os tipos dos argumentos por um tipo genérico, seríamos capazes de implementar um único método**

Métodos Genéricos

- **Se as operações realizadas por vários métodos sobrecarregados forem idênticas para cada tipo de argumento, os métodos sobrecarregados podem ser codificados mais compactamente e convenientemente com um único método genérico**

- Pode-se escrever uma única declaração de método genérico que pode ser chamada com argumentos de tipos diferentes

- Com base nos tipos dos argumentos passados para o método genérico, o compilador trata cada chamada de método apropriadamente

Métodos Genéricos

- Todas as declarações de métodos genéricos contém uma **seção de parâmetros de tipo** delimitada por colchetes angulares ($<$ e $>$) que precede o tipo de retorno do método
- Cada seção de parâmetro de tipo contém um ou mais parâmetros de tipos (também chamados parâmetros de tipo formais), os quais são separados por vírgulas
- Um **parâmetro de tipo**, também conhecido como uma **variável de tipo**, **é um identificador que especifica um nome genérico do tipo**
- **OBSERVAÇÃO:** os parâmetros de tipo podem representar somente tipos por referência

Métodos Genéricos

```
12  public class MetodosGenericos {
13
14      public static void main (String[] args){
15
16          Integer[] arrayInt = {1, 2, 3, 4, 5, 6};
17          Double[] arrayDbl = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7};
18          Character[] arrayCh = {'a', 'b', 'c', 'd', 'e', 'f'};
19
20          System.out.println("Imprimindo array de inteiros:");
21          printArray(arrayInt);
22          System.out.println("Imprimindo array de double:");
23          printArray(arrayDbl);
24          System.out.println("Imprimindo array de caracteres:");
25          printArray(arrayCh);
26      }
27
28      public static <T> void printArray(T[] array){
29          for(int i=0;i<array.length;i++){
30              System.out.print(array[i] + "\t");
31          }
32          System.out.println();
33      }
34
35  }
```

Resultados da Pesquisa | Saída - Teste (run) ×

```
run:
Imprimindo array de inteiros:
1       2       3       4       5       6
Imprimindo array de double:
1.1     2.2     3.3     4.4     5.5     6.6     7.7
Imprimindo array de caracteres:
a       b       c       d       e       f
```

Erasure em tempo de compilação

- Quando o compilador traduz um método genérico em `bytecodes`, ele remove a seção de parâmetros de tipo e substitui os parâmetros de tipo por tipos reais

- Esse processo é conhecido por **erasure**

Métodos que Utilizam um Parâmetro de Tipo como Tipo de Retorno

- Vamos considerar o exemplo de um método genérico em que parâmetros de tipo são utilizados no tipo de retorno e na lista de parâmetros
- Vamos considerar um método genérico `maximum` para determinar e retornar o maior dos seus três argumentos do mesmo tipo
- Vale ressaltar que só é possível comparar dois objetos da mesma classe a classe implementar a interface `Comparable<T>`
- **OBSERVAÇÃO:** todas as classes empacotadoras de tipo para tipos primitivos implementam essa interface

Métodos que Utilizam um Parâmetro de Tipo como Tipo de Retorno

```
12  public class MetodosGenericos {
13
14      public static void main (String[] args){
15
16          System.out.println("Máximo dos inteiros 3, 4, 5: " + maximum(3, 4, 5));
17          System.out.println("Máximo dos double 6.6, 7.7, 8.8: " + maximum(6.6, 8.8, 7.7));
18          System.out.println("Máximo das strings melao, pera, abacate: " + maximum("melao", "pera", "abacate"));
19
20      }
21
22      public static <T extends Comparable<T>> T maximum(T var1, T var2, T var3){
23          T max = var1;
24
25          if(var2.compareTo(max) > 0){
26              max = var2;
27          }
28
29          if(var3.compareTo(max) > 0){
30              max = var3;
31          }
32
33          return max;
34      }
35
36  }
```

Resultados da Pesquisa | Saída - Teste (run) ×

```
run:
Máximo dos inteiros 3, 4, 5: 5
Máximo dos double 6.6, 7.7, 8.8: 8.8
Máximo das strings melao, pera, abacate: pera
```

Sobrecarregando Métodos Genéricos

- **Um método genérico pode ser sobrecarregado**
- Uma classe pode fornecer dois ou mais métodos genéricos que especificam o mesmo nome de método, mas diferentes parâmetros de método
- **Um método genérico também pode ser sobrecarregado por métodos não genéricos**
- Quando o compilador encontra uma chamada de método, ele procura a declaração de método que corresponde mais precisamente ao nome de método e aos tipos de argumentos especificados na chamada $\rightarrow$ métodos genéricos são analisados por último

Classes Genéricas

- O conceito de uma estrutura de dados, como um pilha, pode ser entendido independentemente do tipo de elemento que ela manipula
- **Classes genéricas fornecem um meio de descrever o conceito de uma pilha (ou de qualquer outra classe) de uma maneira independente do tipo**
- **Pode-se então instanciar objetos tipos específicos da classe genérica**
- Essa capacidade fornece uma excelente oportunidade de reutilização de software

Classes Genéricas

- Uma classe Stack genérica, por exemplo, poderia ser a base para criar muitas classes `Stack` lógicas (`Stack` de `Double`, `Stack` de `String`, ...)
- Essas classes são conhecidas como **classes parametrizadas** ou **tipos parametrizados** porque aceitam um ou mais parâmetros
- **OBSERVAÇÃO:** novamente, parâmetros de tipo só representam tipos por referência
- Os identificadores dos tipos genéricos devem ser declarados ao lado do nome da classe entre os sinais < e > e, caso haja mais de um tipo, os tipos devem ser separados por vírgula

Classes Genéricas

```
17
19
20
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
```

```java
public class Pilha<T> {

    private ArrayList<T> elementos;

    Pilha(){
        elementos = new ArrayList<T>();
    }

    public void push(T valor){
        elementos.add(valor);
    }

    public T pop(){
        if(elementos.isEmpty()){
            return null;
        }
        int indice = elementos.size() - 1;
        T value = elementos.get(indice);
        elementos.remove(indice);
        return value;
    }

    public int size(){
        return elementos.size();
    }
}
```

Classes Genéricas

```
13  public class ClassesGenericas {
14
15      public static void main(String[] args){
16
            Pilha<Integer> pilha = new Pilha<Integer>();
18
19          pilha.push(5);
20          pilha.push(15);
21          pilha.push(25);
22          pilha.push(55);
23
24          while(pilha.size() > 0){
25              System.out.println(pilha.pop());
26          }
27
28      }
29
```

Resultados da Pesquisa | Saída - Teste (run) ×

```
run:
55
25
15
5
```

Material Complementar

- Generic Methods

  `https://docs.oracle.com/javase/tutorial/java/generics/methods.html`

- Java Generics Example Tutorial – Generic Method, Class, Interface `http://www.journaldev.com/1663/java-generics-example-method-class-interface`

- Usando Generics em Java
  `http://www.devmedia.com.br/usando-generics-em-java/28981`

Imagem do Dia

Programação Orientada a Objetos
http://lives.ufms.br/moodle/

Rafael Geraldeli Rossi
rafael.g.rossi@ufms.br

Slides baseados em [Deitel and Deitel, 2010]

Referências Bibliográficas I


Deitel, P. and Deitel, H. (2010).
Java: How to Program.
How to program series. Pearson Prentice Hall, 8th edition.