Lição 13

Introdução a *Generics*

Objetivos

Ao final desta lição, o estudante será capaz de:

- Enumerar os benefícios de Generics
- Declarar classes utilizando Generics
- Utilizar Generics limitados
- Declarar métodos utilizando Generics
- Usar coleções Java com Generics

Generics

- Incluídos na Versão 5 do Java

- Problemas com *casting* (conversão de tipos):
  - É um perigo em potencial para uma *ClassCastException*
  - *Torna nossos códigos mais poluídos*
  - *Menos legíveis*
  - *Destrói benefícios de uma linguagem com tipos fortemente definidos*

Generics

- Por que generics?

- Benefícios:
  - Permite que uma única classe trabalhe com uma grande variedade de tipos
  - É uma forma natural de eliminar a necessidade de se fazer *cast*
  - Preserva benefícios da checagem de tipos

Generics

- A remoção do *downcast* não significa poder atribuir qualquer coisa ao valor de retorno do método *get* e eliminar o *typecast* também

- Atribuir outra coisa além de *String* à saída do método get causará um erro de equiparação de tipos em tempo de compilação

Generics

- Passaremos agora para o NetBeans

Declarando Classe Utilizando *Generics*

- Para que o fragmento de código anterior funcione, devemos ter definido uma versão *Generics* da classe *ArrayList*

- A versão mais recente do Java fornece aos usuários versões que utilizam *Generics* para todas as classes *Collection*

Declarando Classe Utilizando *Generics*

- Declaração da classe *BasicGeneric*:

```
class BasicGeneric<A>
```

  - Contém o parâmetro de tipo: `<A>`
  - Indica que a classe declarada é uma classe *Generics*
  - Classe não trabalha com nenhuma referência a um tipo específico

- Declaração do atributo:

```
private A data;
```

  - O atributo *data* é de um tipo *Generic* e depende do tipo de dado com que o objeto *BasicGeneric* for desenvolvido para trabalhar

Declarando Classe Utilizando *Generics*

- Declarando uma instância da classe
  - Deve ser especificado o tipo de referência com qual se vai trabalhar

```
BasicGeneric<String> basicGeneric = new
                    BasicGeneric<String>(data1);

BasicGeneric<Integer> basicGeneric = new
                    BasicGeneric<Integer>(data1);
```

Declarando Classe Utilizando *Generics*

- Declaração do método *getData*:

```
public A getData() {
    return data;
}
```

Declarando Classe Utilizando *Generics*

- Instâncias da classe *BasicGeneric*
  - "presa" ao tipo *String*:

```
BasicGeneric<String> basicGeneric = new
                        BasicGeneric<String>(data1);
String data2 = basicGeneric.getData();
```

  - "presa" ao tipo *Integer*:

```
BasicGeneric<Integer> basicGeneric = new
                        BasicGeneric<Integer>(data1);
Integer data2 = basicGeneric.getData();
```

Generics: Limitação "Primitiva"

- Tipos *Generics* do Java são restritos a tipos de referência (objetos) e não funcionarão com tipos primitivos

```
BasicGeneric<int> basicGeneric =
    new BasicGeneric<int>(data1);
```

- Solução:
  - Encapsular tipos primitivos antes de usá-los
  - Utilizar tipos encapsuladores (*wrapper types*) como argumentos para um tipo *Generics*

Compilando Generics

- Para compilar usando JDK (v. 1.5.0):

```
javac -version -source "1.5" -sourcepath src -d
  classes src/SwapClass.java
```

  onde

  - *src* refere-se à localização do código fonte java
  - *class* refere-se à localização onde o arquivo da classe será gravado
  - Exemplo:

```
javac -version -source "1.5" -sourcepath c:\temp
  -d c:\temp c:/temp/MyFile.java
```

Generics Limitados

- No exemplo anterior:
  - Os parâmetros de tipo da classe *BasicGeneric* podem ser de qualquer tipo de dado de referência

- É possível restringir os tipos em potencial usados em instanciações de uma classe que utiliza generics
  - É possível limitar o conjunto de possíveis tipos utilizados como argumento na instanciação da classe para um conjunto de subtipos de um determinado tipo "amarrado" à classe

Generics Limitados

- Limitando as instanciações de tipo de uma classe:
  - Utilize a palavra-chave *extends* no parâmetro de tipo

```
class ClassName <ParameterName extends ParentClass>
```

Generics Limitados

- Permite uma checagem estática de tipos adicional
  - Garante que toda instanciação de um tipo *generic* respeita os limites (ou restrições) que atribuímos a ele
  - Pode-se chamar, de forma segura, qualquer método encontrado no tipo estático do objeto

- Quando não existir um limite explícito no parâmetro
  - O limite default é *Object*
  - Uma instância não pode invocar métodos que não apareçam na classe *Object*

Declarando Métodos Utilizando *Generics*

- O Java também nos permite declarar um método utilizando *generics*

- Por que generics em métodos?
  - Dependências de tipo entre os argumentos e o valor de retorno são naturalmente genéricas
  - Porém a natureza genérica muda de chamada para chamada ao método ao invés de depender da informação do tipo da classe

Declarando Métodos Utilizando *Generics*

- O Java também utiliza um mecanismo de inferência de tipos
  - Automaticamente infere os tipos dos métodos polimórficos baseado no tipo dos argumentos
  - Diminui o excesso de palavras e a complexidade na invocação do método

- Para construir uma nova instância de *ArrayList<Integer>*, simplesmente teríamos o seguinte comando:

```
Utilities.make(Integer(0));
```

Generics e Coleções

```
public boolean add(E o)
public void clear()
public boolean remove(Object o)
public boolean contains(Object o)
public boolean isEmpty()
public int size()
public Iterator<E> iterator()
public boolean equals(Object o)
public int hashCode()
```

Generics e Coleções

- Passaremos agora para o NetBeans

Sumário

- Por que *Generics*?
- Declarando classe utilizando *Generics*
- *Generics* limitados
- Declarando método utilizando *Generics*
- *Generics* e Coleções

Parceiros

- **Os seguintes parceiros tornaram JEDI™ possível em Língua Portuguesa:**