


PROGRAMAÇÃO ORIENTADA A OBJETOS

Classes e Métodos Genéricos

ROTEIRO

- **Generics**
 - **Classes e Métodos Genéricos**
 - **Herança de Classes Genéricas**
 - **Limitações de Generics**
- 

Generics

- Trata-se de um recurso bem parecido com os templates em C++
- Este conceito permite que tipos (classes e interfaces) sejam parâmetros na definição de classes, interfaces e métodos
- Podemos reusar código para diferentes tipos
- Evita o uso de casting

Generics

- **Tipos Genéricos**

- É uma classe ou interface parametrizada sobre tipos
- Em uma classe **que não usa Generics**, qualquer objeto pode ser utilizado
- O problema é que **em tempo de compilação** não é possível saber que tipo de objeto será passado ou retornado
- Para demonstrar como é um código Java sem Generics, vamos considerar o seguinte exemplo:

Generics

- Tipos Genéricos

```
String result = (String) list.get(0);
```

```
System.out.println(result);
```

- Veja que ao recuperar valores de uma lista era necessário realizar o **cast** para um determinado tipo
 - Isso ocorre porque a interface **List** recebia um Object como parâmetro em seu método add
 - **Lembre-se que em Java todos os tipos estendem Object**
- O código acima compila corretamente. Agora vejamos o próximo exemplo

Generics

- Tipos Genéricos

```
List list = new ArrayList();  
list.add(1);  
list.add("2");  
list.add("3");  
String result = (String) list.get(0);  
System.out.println(result);
```

- O que ocorre no código acima?

- Não é possível ter segurança de qual tipo será retornado pela lista em tempo de compilação e o erro somente será percebido em tempo de execução, como o apresentado abaixo

```
out - Exception in thread "main" java.lang.ClassCastException:  
java.lang.Integer cannot be cast to java.lang.String
```

Generics

- Felizmente com uso de **Generics**, vamos economizar as escritas de códigos e torná-los mais limpos e com possibilidade de reutilização
- A vantagem é que o **Generics** vai servir como parâmetro para a classe em questão e assim podemos utilizar esta “**variável**” em todo o escopo da classe

Generics

- **Declaração e Instanciação de um tipo genérico**
 - Deve especificar qual o tipo desejado
 - É parecido à chamada de um método ou construtor, para qual passamos parâmetros
 - Mas em Generics, o parâmetro é um tipo (classe ou interface)
- **Tipos genéricos não podem receber tipos primitivos**
 - Para isso Java possui classes que representam os tipos primitivos
 - `int` → `Integer`
 - `double` → `Double`

```
Caixa<Integer> integerCaixa = new Caixa<Integer>();  
integerCaixa.set(10);
```


Generics

- Uma classe genérica pode ter muitos parâmetros

```
public interface Par<K, V> {  
    public K getKey();  
    public V getValue();  
}
```

```
public class ParaOrdenado<K, V> implements Par<K, V> {  
    private K key;  
    private V value;  
  
    public ParaOrdenado(K key, V value) {  
        this.key = key;  
        this.value = value;  
    }  
  
    public K getKey() { return key; }  
    public V getValue() { return value; }  
}
```

Generics

- Convenções
 - Um parâmetro de tipo pode ser especificado por qualquer palavra não-chave
 - Porém, por convenção, tipos são definidos com uma única letra
 - *Element (Java Collections)*
 - *K – Key*
 - *N – Number*
 - *T – Type*
 - *V - Value*

Classes e Métodos Genéricos

- Para resolver o problema anterior, o que precisamos fazer?
 - Incluir um parâmetro entre colchetes angulares logo após o nome da classe
 - O tipo T pode ser qualquer tipo não primitivo
- Então, o exemplo ficaria assim:

```
public class MinhaLista<T> {  
    private Object[] elementos = new Object[0];  
    public T get(int indice) {  
        return (T) elementos[indice];  
    }  
    public void adiciona(T elemento) {  
        int posicao = elementos.length + 1;  
        elementos = Arrays.copyOf(elementos, posicao);  
        elementos[posicao] = elemento;  
    }  
}
```

Classes e Métodos Genéricos

- No exemplo anterior, ao longo da classe o tipo T estará disponível e poderá ser utilizado na criação de:
 - Variáveis
 - Retorno de métodos (ex: get)
 - Em parâmetros de métodos (ex: add)
- Isso faz com que MinhaLista possa ser utilizada com qualquer tipo, mas somente um tipo nos elementos, uma vez que a lista tenha sido instanciada

• Exemplo

```
MinhaLista<String> lista = new MinhaLista<>();
```

```
lista.add("1");
```

```
lista.add("2");
```

```
//Não precisa de cast, pois o compilador sabe que é uma  
String
```

```
String resultado = lista.get(0);
```

```
System.out.println(resultado);
```

Classes e Métodos Genéricos

- Não são somente as classes e interfaces que possuem a flexibilidade dos *Generics*, também podemos criar métodos genéricos.
- Métodos Genéricos
 - **Conceito de Generics pode ser amplamente aplicado em métodos, sem que a classe seja genérica (métodos estáticos, construtores, não estáticos)**
 - **A lista de parâmetros de tipo deve aparecer antes do tipo de retorno.**

```
public class ParTeste {  
    public static <K, V> boolean compare(Par<K, V> p1,  
                                         Par<K, V> p2) {  
        return p1.getKey().equals(p2.getKey()) &&  
               p1.getValue().equals(p2.getValue());  
    }  
}
```

Classes e Métodos Genéricos

- Métodos Genéricos
- Utilização

```
Par<Integer, String> p1 = new Par<>(1, "laranja");  
Par<Integer, String> p2 = new Par<>(2, "uva");  
boolean x = ParTeste.<Integer, String>compare(p1, p2);
```

```
Par<Integer, String> p1 = new Par<>(1, "laranja");  
Par<Integer, String> p2 = new Par<>(2, "uva");  
boolean x = ParTeste.<Integer, String>compare(p1, p2);
```

```
Par<Integer, String> p1 = new Par<>(1, "laranja");  
Par<Integer, String> p2 = new Par<>(2, "uava");  
boolean x = ParUtil.compare(p1, p2);
```


Classes e Métodos Genéricos

- É importante frisar que quando utilizado em métodos, o tipo genérico pertence ao escopo daquele método
- Não é possível utilizar o tipo em outros métodos
 - Se a classe também é genérica e possui um identificador de tipo igual ao do método, o do método vai sobrescrever o da classe

Herança de Classes Genéricas

- Quando criamos por exemplo, uma caixa de números, qualquer tipo numérico poderá ser armazenado

```
Caixa<Number> caixa = new Caixa<Number>();  
caixa.add(new Integer(10)); // OK  
caixa.add(new Double(10.1)); // OK
```

- **No entanto, no método a seguir**

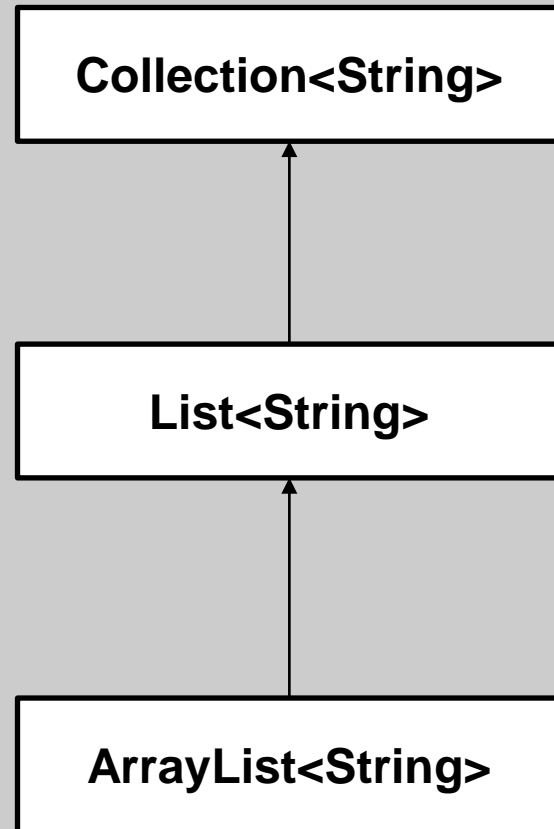
```
public void caixaTest(Caixa<Number> n) { /* ... */ }
```

- Como não há relação de herança entre as classes não podemos passar `Caixa<Integer>` ou `Caixa<Double>`

Herança de Classes Genéricas

- É possível herdar ou implementar classes genéricas
 - Classe filha pode ou não ser genérica
 - Toda a relação entre os tipos genéricos da classe mãe e da filha é definida na declaração de classe
- **Por exemplo**
 - ArrayList<E> implementa List<E>
 - List<E> herda de Collection<E>
- A relação de herança **somente quando os tipos são iguais**

Herança de Classes Genéricas



Limitações de Generics

- Não é possível instanciar um tipo genérico utilizando tipos primitivos

```
Par<int, char> p = new Par<>(2, 'b'); // erro de compilação  
Par<Integer, Character> p = new Par<>(2, 'b'); // OK
```

- Não é possível criar instâncias de parâmetros genéricos

```
public static <E> void append(List<E> list) {  
    E elem = new E(); // erro de compilação  
    list.add(elem);  
}
```

Referências

1. **Java Como Programar: Paul Deitel & Harvey Deitel - 10ª Edição**
2. **Java Como Programar: Paul Deitel & Harvey Deitel - 8ª Edição**

PROGRAMAÇÃO ORIENTADA A OBJETOS

Classes e Métodos Genéricos