Tipos Genéricos

Maurício Linhares

Conteúdo da Aula

- Segurança de tipos em tempo de compilação;
- O que são tipos genéricos;
- Criando métodos genéricos;
- Herança e sobrecarga de métodos genéricos;

Motivações para os tipos genéricos

- Diminuir a quantidade de operações de "cast" nos programas;
- Oferecer uma checagem em tempo de compilação dos tipos utilizados em operações polimórficas;
- Garantir que coleções sempre contém apenas um tipo de objeto;

Código genérico é mais expressivo

```
List lista = new ArrayList();

lista.add( 20 );
lista.add( 22.00);
```

Essa lista é de que?

```
List <Integer> inteiros =
                        new ArrayList<Integer>();
```

Esta é uma lista de números inteiros

Código genérico é mais seguro

```
List<String> strings = new
                                    ArrayList<String>();

strings.add( 20 ); //Erro de compilação
```

Métodos genéricos e classes genéricas

Oferecem meios pelos quais uma única declaração de método ou classe se relacione com vários tipos de objetos diferentes transparentemente, sem que o código seja reescrito

Sobrecarga de métodos

Sobrecarga de método é a implementação de um método já existente no objeto mudando seus parâmetros ou o seu retorno (apenas em retornos covariantes).

O compilador procura o método que mais se assemelha com o método declarado no código, através do objeto onde ele está sendo chamado e os parâmetros passados.

Exemplo de sobrecarga de métodos

```
public static void imprimirArray( String[] strings ) {

        for (String string : strings) {
                System.out.printf("%s\n ", string);
        }
}

public static void imprimirArray( Character[] chars ) {

        for (Character character : chars) {
                System.out.printf("%s\n ", character);
        }
}
```

Chamando os métodos sobrecarregados

```
String[] stringArray = { "José", "Carlos"};
Character[] charArray = { 'm', 'a', 'r' };

imprimirArray(charArray);

imprimirArray(stringArray);
```

Simplificando com genéricos

O mesmo código se repete para os dois tipos, então ele pode ser simplificado com o uso de genéricos.

Reformulando o método usando genéricos

```
public static <A> void imprimirArray
        ( A[] array ) {

   for (A elemento : array) {
        System.out.printf("%s\n ", elemento);
   }

}
```

Chamando os métodos genéricos

```
String[] stringArray = { "José", "Carlos"};
Character[] charArray = { 'm', 'a', 'r' };

imprimirArray( charArray );

imprimirArray( stringArray );
```

O que aconteceu?

O parâmetro de tipo <A> declarado na versão genérica do método imprimirArray() foi trocado em tempo de compilação pelo tipo do objeto passado como parâmetro.

Tradução de tipos

```
public static <A> void imprimirArray ( A[] array ) {
   for (A elemento : array) {
         System.out.printf("%s\n ", elemento);
   }
}

public static void imprimirArray( Character[] chars ) {
         for (Character character : chars) {
               System.out.printf("%s\n ", character);
         }
}
```

Declarando um método genérico

- Métodos genéricos declaram os parâmetros de tipo (<A>) antes da declaração de tipo de retorno;
- Os parâmetros de tipo podem ser reutilizados em qualquer lugar dentro do corpo do método;

Parâmetros de tipo

- São as declarações que definem os comportamentos genéricos, guardando tipos de objetos;
- São sempre definidos entre os símbolos de < (menor) e > (maior);
- É uma boa prática defini-los com uma única letra maiúscula (como em <A>);
- Uma única declaração pode conter vários parâmetros separados por vírgulas (<K,V>)

Revendo o código

```
public static void imprimirArray
   ( Object[] objects ) {

        for (Object object : objects) {
                System.out.printf("%s\n ",object);
        }

}
```

O código polimórfico funciona da mesma maneira

Foi possível trocar todo o código genérico por uma implementação polimórfica que funciona da mesma maneira.

Parametros de tipo como retorno do método

Além de ser utilizados dentro do corpo do método, os parâmetros de tipo podem ser utilizados como o tipo de retorno dos métodos nos quais eles são declarados.

A interface genérica Comparable<T>

- É utilizada para fazer a comparação de objetos;

- Define um único método genérico "compareTo (T o)" que retorna um inteiro;

A interface genérica Comparable<T>

- O método "compareTo()" recebe um objeto do tipo <T> e deve retornar um inteiro negativo se o objeto passado como parâmetro for maior;

- Zero (0) se os objetos forem equivalentes;

- Um inteiro positivo se o objeto passado como parâmetro for menor;

Exemplo de Comparable<Integer>

```
Integer menor = 10;
Integer maior = 20;

System.out.println( menor.compareTo(maior) );
// Imprime -1
System.out.println( maior.compareTo(maior) );
// Imprime 0
System.out.println( maior.compareTo(menor) );
// Imprime 1
```

Métodos com retorno genérico

```
public static <T extends Comparable<T> > T maior (T x, T
   y) {

   if ( x.compareTo( y )  > 0 ) {
         return x;
   }
   return y;
}
```

Exemplo da chamada de método

```
Integer dez = 10;
Integer vinte = 20;
Integer resultado = maior (vinte, dez);

out.printf("\nO maior dos números { %d, %d } é %d\n", dez,
   vinte , resultado );
```

Tradução do compilador

```
public static Integer
                    maior (Integer x, Integer y)
{
   if ( x.compareTo( y )  > 0 ) {
         return x;
   }
   return y;
}
```

“extends” em um parâmetro de tipo

- Indica a mesma relação de herança entre objetos, tanto para interfaces quanto para classes;
- Indica o “limite superior” dos objetos parametrizáveis, que neste caso são objetos que implementam a interface Comparable<T>, objetos que não sejam do tipo do “limite superior” não podem ser utilizados;

Exemplo de limite superior usando “extends”

Sobrecarregando métodos genéricos

- Métodos genéricos podem ser sobrecarregados normalmente por métodos não genéricos;
- É possível sobrecarregar um método genérico com outro método genérico, contanto que os parâmetros sejam diferentes ou mais específicos;

Sobrecarga com método não genérico

```
public static <A extends List<?>> void imprimirArray( A []
   array ) {
   for (A elemento : array) {
         out.printf("%s\n ", elemento);
   }
}

public static void imprimirArray (String[] strings)  {
   for (String string : strings) {
         out.printf( "Valor: %s", string );
   }
}
```

Sobrecarga com método genérico

```
public static <A> void imprimirArray( A[] array ) {
    for (A elemento : array) {
            out.printf("%s\n ", elemento);
    }
}

public static <A extends Number> void imprimirArray( A[] array ) {
    for (A elemento : array) {
            out.printf("%s\n ", elemento);
    }
}
```

Herança de métodos genéricos

O comportamento de herança não se altera com métodos genéricos, a subclasse que herdar um método genérico vai poder utilizá-lo normalmente como se fosse um método comum herdado de sua superclasse.

Revisão – introdução aos tipos genéricos

- Quais as vantagens de uso dos tipos genéricos?
- O que é sobrecarga de método?
- Como o compilador encontra o método que vai chamar?

Revisão – parâmetros de tipo

- O que é um parâmetro de tipo?
- Como um parâmetro de tipo é declarado em um método?
- Onde ele pode ser utilizado dentro do método?
- Como declarar vários parâmetros de tipo?
- Como é feita a tradução dos parâmetros de tipo?

Revisão – interface Comparable<T>

- Para quê a interface Comparable<T> é utilizada?
- Qual o comportamento esperado do método "compareTo()"?

Revisão – retorno genérico e limite superior

- Como definir um tipo de retorno genérico?
- O que ocorre na tradução do retorno genérico para o retorno normal?
- O que é o "limite superior"?
- A palavra chave "extends" pode ser utilizada para quais tipos em uma declaração genérica?
- Qual é o limite superior em <T extends Cloneable>?

Revisão – sobrecarga de métodos genéricos

- Como sobrecarregar um método genérico com um método normal?

- Como sobrecarregar um método genérico com outro método genérico?

Mais informações

- JavaDoc do Java SE 1.5 - http://java.sun.com/j2se/1.5.0/docs/api/
- DEITEL, H. M.; DEITEL, P. J.; **Java Como Programar 6ª Edição**. Editora Campus, 2005.
- Grupo de Usuários Java – http://www.guj.com.br/

Laboratório – Sobrecarga de métodos

- Criar um método que leia um array de Strings e junte todos eles em uma única String, separados por um "\n";
- Sobrecarregar o mesmo método com um array de objetos Date;
- Sobrecarregar o mesmo método com objetos Locale;

Laboratório – parâmetros de tipo

- Desenvolver um método genérico que faça o o mesmo que o do exercício anterior faz em outra classe;
- Traduzir a chamada genérica em um método polimórfico;
- Criar um método que receba dois parâmetros diferentes com parâmetros de tipos diferentes e os coloque como chave e valor em uma coleção HashMap;

Laboratório – retorno genérico

- Desenvolver um método que faça uma cópia do objeto passado como parâmetro (usando o método "clone()") e retorne o objeto clonado;
- Desenvolver uma tradução para o método genérico;

Revisão – sobrecarga de métodos

- Desenvolver uma versão genérica sobrecarregada do método anterior que só aceite objetos que implementam a interface Cloneable;
- Desenvolver uma versão sobrecarregada desse mesmo método apenas para objetos Date;

Mais informações

- JavaDoc do Java SE 1.5 - http://java.sun.com/j2se/1.5.0/docs/api/
- DEITEL, H. M.; DEITEL, P. J.; **Java Como Programar 6ª Edição**. Editora Campus, 2005.
- Grupo de Usuários Java – http://www.guj.com.br/