AULA 2 – USANDO GENERICS

GSI020 – Programação Orientada a Objetos II

Prof. Dr. Murillo G. Carneiro
mgcarneiro@ufu.br

UFU

Material baseado nos slides cedidos pelo Prof. Rafael D. Araújo (FACOM/UFU)

Objetivo da aula

- Entender e implementar classes e métodos com tipos genéricos em Java.

Motivação

- Métodos sobrecarregados são frequentemente utilizados para realizar operações semelhantes em tipos diferentes de dados.
- Imagine um método `imprimeArray` que imprime dos valores de um array de inteiros, um array de double e um array de caracteres.
- Qual seria a diferença de implementação entre eles?

Exemplo sem generics

```
public static void imprimeArray(Integer[] inputArray) {
    for (Integer element : inputArray)
        System.out.printf("%s ", element);
    System.out.println();
}

public static void imprimeArray(Double[] inputArray) {
    for (Double element : inputArray)
        System.out.printf("%s ", element);
    System.out.println();
}

public static void imprimeArray(Character[] inputArray) {
    for (Character element : inputArray)
        System.out.printf("%s ", element);
    System.out.println();
}
```

Exemplo sem generics

```
Integer[] intArray = {1, 2, 3, 4, 5};
Double[] doubleArray = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7};
Character[] charArray = {'H', 'E', 'L', 'L', 'O'};

imprimeArray(intArray);
imprimeArray(doubleArray);
imprimeArray(charArray);
```

Tipos primitivos x por referência

- Qual a diferença entre tipo int e Integer?
- Tipos primitivos ou por referência
- Classes empacotadoras de tipo

somente tipos por referência podem ser
usados para especificar os tipos genéricos em métodos e classes genéricos

Motivação

- Seria mais conveniente implementar o método imprimeArray apenas uma vez e o tipo dos dados do array fosse detectado em tempo de compilação.
- Com isso, seria mais fácil compreendê-lo (legibilidade) e, ainda, seria possível reutilizá-lo (reusabilidade) em outras aplicações.

Exemplo com generics

- Basta substituir os 3 métodos, por:

```
public static <T> void imprimeArray(T[] inputArray) {
    for (T element : inputArray)
        System.out.printf("%s ", element);
    System.out.println();
}
```

Métodos genéricos

- Os parâmetros, variáveis locais e o tipo de retorno podem ser definidos na chamada do método.
- Permite ao mesmo método ser invocado usando-se tipos distintos (sem precisar sobrescrevê-lo).

Tipo genérico T

- É válido somente para tipos referenciáveis (objetos) e não podem ser definidos para tipos primitivos (como int, double, char, etc.)
- Permite segurança de tipos em *tempo de compilação*
  - *o compilador determina se as operações no corpo do método podem ser aplicadas a elementos do tipo utilizado na chamada*

Tipo genérico T

- Por convenção, parâmetros de tipos genéricos são nomeados por letras maiúsculas únicas
- Nomes mais comuns:
  - *T (Type)*
  - *K (Key)*
  - *V (Value)*
  - *E (Element)*

Classe genérica

- O conceito de *Generics* pode ser estendido às classes
- Os atributos da classe podem ser definidos no momento da instanciação do objeto
- Essas classes são conhecidas como classes parametrizadas ou tipos parametrizados
- Recurso útil ao definir classes como estruturas de dados
  - *Exemplo: uma pilha pode ser de elementos do tipo Integer, String, Double, etc.*

Exemplo - Pilha

```
public class Pilha <E> {

    private int tamanho;
    private E[] elementos;

    public Pilha (int tam) {
        this.tamanho = 0;
    }
}
```

```
// pilha de objetos Double
Pilha<Double> doubleStack = new Pilha<Double>(5);

// pilha de objetos Integer
Pilha<Integer> integerStack = new Pilha<Integer>(5);
```

Ao instanciar um objeto do tipo genérico, é necessário definir um tipo concreto.

Curingas (*wildcards*)

- Curinga é o nome dado ao identificador ? em códigos genéricos.
- Representa um tipo desconhecido que pode ser utilizado em algumas situações como um tipo de parâmetro.

Curingas (*wildcards*)

- Os curingas permitem especificar parâmetros de método, valores de retorno, variáveis ou campos e assim por diante, que atuam como supertipos ou subtipos de tipos parametrizados.

Curingas (*wildcards*)

- Exemplo:

Curinga indicando que esse método recebe uma lista de quaisquer objetos do tipo Animal.

"a própria classe ou uma subclasse de"

```
public static void imprimeAnimais(List<? extends Animal> animais) {
    for (Animal animal : animais) {
        System.out.println(animal.getTipoAnimal());
    }
}
```

Uma outra opção seria utilizar List<? **super** Cachorro> que indica que os objetos da lista podem ser do tipo da "própria classe ou uma superclasse de".

Exercício 1

- Implemente o exemplo de impressão dos arrays (slides 4 e 5) usando método genérico (slide 8).

Exercício 2

- Implemente um método para calcular o maior valor dentre 3 números do mesmo tipo (Integer ou Float ou Double)
  - *Dica 1: o tipo de retorno de um método genérico também pode ser T*
  - *Dica 2: operador relacional “>” não pode ser utilizado com tipos por referência. No entanto, é possível comparar dois objetos da mesma classe (utilizando o método compareTo) se essa classe implementar a interface genérica Comparable<T> (do pacote java.lang)*

Referências

- DEITEL, H. M. Java: como programar, 8. ed. São Paulo: Prentice Hall, 2010. Capítulo 16 e 20.