Universidade Estadual do Centro Oeste - UNICENTRO
Departamento de Ciência da Computação - DECOMP

Programação Orientada à Objetos II
Tipos Genéricos – Aula I

Professora: Inali Wisniewski Soares

Tipos Genéricos (TG)

- Permitem criar interfaces, classes e métodos nos quais o tipo de dado com o qual operam é especificado como parâmetro.

- Código genérico funciona automaticamente com o tipo de dado passado para o seu parâmetro de tipo.

- Muitos algoritmos são logicamente iguais, independente do tipo de dado ao qual estão sendo aplicados.
  - Exemplo: mecanismo que dá suporte a uma pilha é o mesmo sem importar se ela está armazenando itens de tipo Integer, String ou Object.

Motivações para uso de TG

- Evita trabalho extra quando são inseridos novos tipos de objetos em um algoritmo.

- Programação genérica permite abstrair os tipos de objetos.

Parâmetros de tipos

- Identificadores que especificam o nome de um TG.
- Por convenção, são nomeados por letras maiúsculas únicas.
  - para distinguir parâmetros de tipos de nomes de interfaces, classes e métodos.
- Usos mais comuns:
  - E - Element (uso comum no *Java Collections Framework*)
  - K - Key
  - N - Number
  - T - Type
  - V - Value

Forma geral de uma classe genérica

- Sintaxe de declaração de uma classe genérica:

```
public class nome-classe <lista-parâm-tipo> {
    // ...
}
```

Exemplo1 - classe genérica simples (1)

```
// Classe genérica simples.
// Aqui, T é um parâmetro de tipo que será substituído pelo
// tipo real quando um objeto de tipo Gen for criado.
public class Gen<T> {
  T ob; // declara uma referência a um objeto de tipo T

  // passa para o construtor uma referência
  // a um objeto de tipo T.
  public Gen(T o) {
    ob = o;
  }

  // retorna ob.
  public T getob() {
    return ob;
  }

  // O método showType() exibe o tipo de T.
  // Ele faz isso chamando getName() no objeto Class
  // retornado pela chamada a getClass() em ob
  public void showType() {
    System.out.println("Tipo de T = " + ob.getClass().getName());
  }
}
```

Declara uma **classe genérica**. **T** é o nome de um parâmetro de **TG**. **T** está dentro de <> (colchetes angulares).

A classe **Object** define o método **getClass()**. Assim, o método **getClass()** é membro de todos os tipos de classe. Ele retorna um objeto **Class** correspondente ao tipo de classe do objeto em que foi chamado. **Class** é uma classe definida dentro de **java.lang** que encapsula informações sobre outra classe. Ela define vários métodos que podem ser usados na obtenção de informações sobre uma classe no tempo de execução.

Entre eles, está o método **getName()**, que retorna uma representação do nome da classe na forma de string.

Exemplo1 - classe genérica simples (2)

```
public class TesteGen {
    public static void main(String[] args) {
        Gen<Integer> iOb;
        Gen<String> strOb;

        // Cria um objeto Gen<Integer> e atribui sua
        // referencia para iOb. Note o uso de autoboxing
        // para encapsular o valor 88 em um objeto Integer.
        iOb = new Gen<Integer>(88);

        iOb.showType();

        // Obtém o valor em iOb. Note que
        // nenhuma coerção (cast) é necessário.
        int v = iOb.getOb();
        System.out.println("valor: " + v);

        strOb = new Gen<String>("Testa Genericos");
        strOb.showType();
        String str = strOb.getOb();
        System.out.println("valor: " + str);
    }
}
```

Cria uma referência a um objeto de tipo Gen<Integer>.

Instancia um objeto de tipo Gen<Integer>.

Saída produzida pelo programa:
Tipo de T = java.lang.Integer
valor: 88

Tipo de T = java.lang.String
valor: Testa Genericos

TG só funcionam com objetos

- Na declaração de uma instância de um TG, o argumento de tipo passado para o parâmetro deve ser um tipo de referência (objeto).
  - Não pode ser usado um tipo primitivo, tal como int ou char.

- Exemplo de instrução **válida**:

```
Gen<Integer> iOb = new Gen<Integer>(53);
```

- Exemplo de instrução inválida:

```
Gen<int> iOb = new Gen<int>(53);
```

Tipos Genéricos e parâmetros de tipos

- TG diferem de acordo com seus parâmetros de tipo.
- Exemplos:

```
Gen<Integer> iOb = new Gen<Integer>(53);
Gen<Double>  dOb = new Gen<Double> (3.2);
iOb = dOb // Errado!
```

- **iOb** e **dOB** são **TG**, porém, os mesmos são incompatíveis, pois são referências a tipos diferentes já que seus parâmetros de tipos diferem.
  - Portanto TG adicionam segurança de tipos e evitam erros.

Exemplo 2 - Classe genérica aprimorada

- Podem ser declarados dois ou mais parâmetros de tipos em um TG.

```
public class DoisGen <T,V>{
  T ob1;
  V ob2;

  DoisGen(T o1, V o2) {
    ob1 = o1;
    ob2 = o2;
  }
  public T getOb1() {
    return ob1;
  }
  public V getOb2() {
    return ob2;
  }
}
// Criar objetos do tipo DoisGen <T,V>
//DoisGen<String,Integer> par = new DoisGen<String,Integer>("Teste", 88);
```

Referências

- DEITEL, P; DEITEL, H. Java como programar. São Paulo: 8ª edição. Pearson Education do Brasil, 2010.
- HORSTMANN, C. S.; CORNELL, G. Core Java, volume I – fundamentos. 8ª edição. São Paulo: Pearson Education do Brasil, 2010.
- SHILDT, H; SKRIEN, D. Programação com Java – Uma Introdução Abrangente. Porto Alegre: AMGH, 2013.