Polimorfismo paramétrico

Prof. Pedro Pongelupe

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Departamento de Ciência da Computação

Sumário

1. Polimorfismo universal paramétrico
2. Generics em Java
   - Generics: Classes
   - Generics: variáveis e arranjos
   - Generics: interfaces
   - Generics: métodos
   - Generics: herança

Polimorfismo universal paramétrico: *Generics*

- Funções ou tipos abstratos que operam da mesma forma sobre objetos de tipos diferentes (funções ou tipos genéricos).
  - Trata valores de forma idêntica, sem depender do conhecimento sobre seus tipos.

- E as funções genéricas baseadas na classe Object?
  - Código se baseia no polimorfismo de inclusão. Para se usar funções específicas do objeto é necessário fazer *type casting* (como no *Java Collections v1.4* ou anterior).
  - Não era *type safe* (não previne erros de tipos) .

Polimorfismo universal paramétrico: *Generics*

- Exemplo de Coleções no *Java Collections Framework v1.4*:

```
ArrayList listaDeProfessores = new ArrayList();

listaDeProfessores.add("Pedro Lopes");
     // Adiciona um professor na lista
listaDeProfessores.add("Zé da Silva");
     // Adiciona um professor na lista

String prof1 = (String) listaDeProfessores.get(0);
     // Recupera o 1o professor da lista

Professor prof2 = (Professor) listaDeProfessores.get(1);
     // Essa última linha irá compilar, uma vez que a função
     // get(n) retorna Object, mas irá produzir erro de execução.
```

Polimorfismo paramétrico: Tipos como parâmetros

Listas com tipos parametrizados

List<Tipo> variavel = **new** ArrayList<Tipo>();

- A classe ArrayList aceita um tipo como parâmetro:
  - Tipo passado como parâmetro usando < >.
  - Por compatibilidade: Versão antiga funciona, mas produz *warnings*.

```
ArrayList<String> listaDeProfessores = new ArrayList();
    // Define tipo da lista como parâmetro

listaDeProfessores.add("Pedro Lopes");
listaDeProfessores.add("Zé da Silva");

String prof1 = listaDeProfessores.get(0);
    // Sem type casting
Professor prof2 = (Professor) listaDeProfessores.get(1);
    // Produz erro de compilação (erro de tipo)
```

Generics: Classes

Classes parametrizadas

public class Nome<Tipo> { }

ou

public class Nome<Tipo1, Tipo2, ..., TipoN> { }

- Um tipo deve ser passado como parâmetro para <Tipo> no momento da construção do objeto.
- o resto da sua classe pode ser implementada baseada nesse nome de tipo.
  - Convenção de nomenclatura – usar apenas um caractere:

    T para Tipo, E para Elemento, N para Número, K para Chave – *Key*, e V para Valor.

Generics: variáveis e arranjos

- Não é possível se construir objetos ou arranjos com tipos parametrizados.

```
public class Shulambs<T> {
  private T campo;                  // ok
  private T[] arranjo;              // ok

  public Shulambs(T param) {
        campo = new T();            // erro
        arranjo = new T[10];        // erro
  }
}
```

Generics: variáveis e arranjos

- Pode-se criar variáveis e passar parâmetros.
- Pode-se fazer *type casting* de arranjos a partir de Object[].

```
public class ShulambsFixed<T> {
    private T campo;
    private T[] arranjo;

    @SuppressWarnings("unchecked")
    public ShulambsFixed(T param) {
        campo = param;
        arranjo = (T[]) new Object[10];
    }
}
```

Generics: comparação de objetos

- Generics usam semântica de referência.
- Deve-se comparar objetos de tipos parametrizados usando o método T.equals(T).

```
public class ArrayList<E> {
  ...
  public int indexOf(E value) {
    for (int i = 0; i < size; i++) {
    // if (elementData[i] == value) {
       if (elementData[i].equals(value)) {
          return i;
       }
    }
    return -1;
  }
}
```

Generics: interfaces

```
// Representa uma lista de valores
public interface List<E> {
    public void add(E value);
    public void add(int index, E value);
    public E get(int index);
    public int indexOf(E value);
    public boolean isEmpty();
    public void remove(int index);
    public void set(int index, E value);
    public int size();
}

public class ArrayList<E> implements List<E> { ... }

public class LinkedList<E> implements List<E> { ... }
```

Generics: métodos

- Para tornar apenas um método genérico, o tipo de retorno deve ser precedido pelo parâmetro de tipo.

```
public class Collections {
    ...
    public static <T> void copy(List<T> dst, List<T> src) {
        for (T t : src) {
            dst.add(t);
        }
    }
}
```

Generics: Herança

- É possível determinar heranças quando utilizamos a palavra-chave *extends*.

```
public class FechadorRecurso {

        public static <T extends Closeable> void
    closeAll(T... fechaveis)    throws IOException {
                for (T t : fechaveis) {
                        t.close();
                }
        }
}
```

Obrigado!!

Muito obrigado pela atenção! Alguma dúvida? Bora praticar!!!

"*A leitura do mundo precede a leitura da palavra.*"