Java Avançado

Generics

Softblue
cursos online

Tópicos Abordados

- Introdução
- Comparação entre usar ou não generics
- Definição de classes que usam generics
- Generics e as subclasses
- Wildcards
  - *extends*
  - *super*
- Definição de métodos que usam generics

Introdução

- O Generics foi introduzido a partir do Java 5
- Similar aos templates em C++
- Permitem a parametrização de tipos de dados

Sem generics X com generics

Sem generics

```
List strings = new ArrayList();
strings.add("abc");
strings.add("def");
strings.add(new Integer(1));

String s = (String) strings.get(0);
```

Com generics

```
List<String> strings = new ArrayList<String>();
strings.add("abc");
strings.add("def");
strings.add(new Integer(1));

String s = strings.get(0);
```

Não compila porque o objeto não é uma *String*

Definindo uma classe com generics

```
public class Gerenciador {

    private Object obj;

    public void setObjeto(Object obj) {
        this.obj = obj;
    }

    public Object getObjeto() {
        return obj;
    }
}
```

O uso de *Object* possibilita que qualquer objeto seja utilizado

```
Casa c = new Casa();
Gerenciador g = new Gerenciador();

g.setObjeto(c);
c = (Casa) g.getObjeto();
```

Definindo uma classe com generics

```
public class Gerenciador<T> {

    private T obj;

    public void setObjeto(T obj) {
        this.obj = obj;
    }

    public T getObjeto() {
        return obj;
    }
}
```

Agora o tipo do dado é parametrizável

```
Casa c = new Casa();
Gerenciador<Casa> g = new Gerenciador<Casa>();

g.setObjeto(c);
c = g.getObjeto();
```

Apenas uma instância de *Casa* pode ser fornecida

O casting não é necessário

Generics e as subclasses

- Suponha que existem as classes
  - *Animal*
  - *Cachorro* (estende de *Animal*)
- O código abaixo funciona normalmente

```
Cachorro[] cachorros = new Cachorro[2];
cachorros[0] = new Cachorro();
cachorros[1] = new Cachorro();

Animal[] animais = cachorros;
```

É possível atribuir um array de *Cachorro* a um array de *Animal* devido à herança

Generics e as subclasses

- E o mesmo código usando generics?

```
List<Cachorro> cachorros = new ArrayList<Cachorro>();
cachorros.add(new Cachorro());
cachorros.add(new Cachorro());

List<Animal> animais = cachorros;
```

Não é possível atribuir, mesmo sendo uma lista da subclasse

Em generics, a atribuição só pode ser feita se o tipo parametrizado é o mesmo

Wildcard

- Utilizado para deixar em aberto o tipo parametrizado
- Representado por "**?**"

```
public void imprimir(Collection<?> c) {
    for(Object o : c) {
        System.out.println(o);
    }
}
```

A coleção pode ser de qualquer tipo

Wildcard e o *extends*

- Quando é necessário deixar o tipo do dado em aberto mas garantir que este tipo deve estender de determinada classe, é possível usar o *extends*

```
public void imprimir(Collection<? extends Animal> c) {
    for(Animal a : c) {
        a.andar();
    }
}
```

A coleção pode ser de qualquer tipo que seja uma subclasse de *Animal* ou a própria classe *Animal*

Wildcard e o extends

- Quando a restrição deve ser feita a classes que implementam uma interface, o *extends* também é usado

```
public void imprimir(Collection<? extends Comparable> c) {
    //...
}
```

Comparable é uma interface, mas o *extends* é usado mesmo assim

Quando o *extends* é usado, não é possível adicionar elementos

Wildcard e o *super*

- Quando é necessário deixar o tipo do dado em aberto mas garantir que este tipo deve ter uma determinada superclasse, é possível usar o *super*

```
public void imprimir(Collection<? super Cachorro> c) {
    //...
}
```

A coleção pode ser de qualquer tipo que seja uma superclasse de *Cachorro* ou a própria classe *Cachorro*

Quando o *super* é usado, a coleção pode receber novos elementos

Definindo métodos com generics

- Além do tipo parametrizado poder ser definido a nível de classe, ele também pode ser definido a nível de método

T é o tipo parametrizado

```
private <T> List<T> criarLista(T e) {
    List<T> l = new ArrayList<T>();
    l.add(e);
    return l;
}
```

```
List<String> l1 = criarLista("a");
List<Integer> l2 = criarLista(2);
```

A lista é criada e retornada com o tipo do parâmetro