UNIRIO

Programação Genérica

Aula 20

BSI — 2018.2

Jefferson Elbert Simões

CCET/DIA

6 de novembro de 2018

Previously on TP2...

- Fundamentos: classes, objetos, atributos, métodos
- Referências
- Construtores e destrutores
- Métodos e atributos estáticos
- Encapsulamento e visibilidade
- Tipos de enumeração
- Agregação
- Herança
- Polimorfismo: sobrecarga, sobreposição
- Classes e métodos abstratos
- Interfaces
- Final
- Exceções

Programação genérica

"Generic programming is a programming method based in finding the most abstract representation of algorithms. That is, you start with an algorithm and find the most general set of requirements that allows it to perform and to perform efficiently. The amazing thing is that many different algorithms need the same set of requirements and there are multiple implementations of these requirements."

(Alexander Stepanov)

Programação genérica

- Programação genérica é a ideia de escrever código sem depender dos tipos das variáveis/objetos envolvidos
  - Em muitos casos, apenas a mudança de um tipo exige a escrita de múltiplos trechos de código
  - Exemplo: função para buscar um elemento um array (de `int`'s? `float`'s? `Aluno`'s?)
- As principais bibliotecas do Java utilizam programação genérica
  - Estruturas de dados (ex: `ArrayList`), modelos de estruturas (ex: `List`) e algoritmos (ex: `Collections.sort`)

Programação genérica

```
// retorna a primeira posição de array em que o
// elemento alvo se encontra, -1 se não for encontrado
public static int busca( int [] array, int alvo ) {
    for( int pos = 0; pos < array.length - 1; pos++ )
        if( array[pos] == alvo )
            return pos;
    return -1;
}
```

Programação genérica

```
// retorna a primeira posição de array em que o
// elemento alvo se encontra, -1 se não for encontrado
public static int busca( long [] array, long alvo ) {
    for( int pos = 0; pos < array.length - 1; pos++ )
        if( array[pos] == alvo )
            return pos;
    return -1;
}
```

Programação genérica

```
// retorna a primeira posição de array em que o
// elemento alvo se encontra, -1 se não for encontrado
public static int busca( float [] array, float alvo )
{
    for( int pos = 0; pos < array.length - 1; pos++ )
        if( array[pos] == alvo )
            return pos;
    return -1;
}
```

Programação genérica

Três tipos, três implementações, 3x mais código

```
// retorna a primeira posição de array em que o
// elemento alvo se encontra, -1 se não for encontrado
public static int busca( float [] array, float alvo )
{
    for( int pos = 0; pos < array.length - 1; pos++ )
        if( array[pos] == alvo )
            return pos;
    return -1;
}
```

Métodos genéricos

Métodos parametrizados por um tipo

- Tipo deve sempre ser uma classe (ex: Integer)
- Typecast para tipos primitivos

```
// retorna a primeira posição de array em que o
// elemento alvo se encontra, -1 se não for encontrado
public static <T> int busca( T [] array, T alvo ) {
    for( int pos = 0; pos < array.length - 1; pos++ )
        if( array[pos] == alvo )
            return pos;
    return -1;
}
Integer [] arrayTeste = {5, 1, 2, 9, 4, 7};
int alvoTeste = 1;
System.out.println( busca( arrayTeste, alvoTeste ) );
```

Tipos genéricos

Classes/interfaces parametrizadas por um tipo

- Tipo deve ser escolhido na instanciação
- Deve ser não-primitivo (ex: classe, interface, array)

```
public class Box<T> {
    private T object;

    public void set( T newObj ) { object = newObj; }
    public T get() { return object; }
}

Box<Integer> minhaCaixa = new Box<>();
minhaCaixa.set( 5 );
```

Parâmetros limitados

É possível restringir os tipos aceitos, exigindo que sejam filhos de uma classe ou implementem uma interface dada:

```
public class Box<T extends Pessoa> {
    private T object;

    public void set( T newObj ) { object = newObj; }
    public String getNome() {
        return object.getNome();
    }
}

Box<Aluno> caixaAluno = new Box<>();
~~Box<Integer> caixaInteiro = new Box<>();~~
```

Parâmetros limitados

É possível restringir os tipos aceitos, exigindo que sejam filhos de uma classe ou implementem uma interface dada:

```
public class Box<T extends NumeroComModulo> {
    private T object;

    public void set( T newObj ) { object = newObj; }
    public String getFormaPadrao() {
        return object.formaPadrao();
    }
}

Box<Racional> caixaRacional = new Box<>();
~~Box<Integer> caixaInteiro = new Box<>();~~
```

Herança

Herança continua valendo para os objetos utilizados por classes genéricas:

```
public class Box<T> {
    private T object;

    public void set( T newObj ) { object = newObj; }
    public T get() {
        return object;
    }
}
```

```
Box<Pessoa> caixaAluno = new Box<>();
caixaAluno.set(new Aluno(...));
```

Herança

E também vale para os tipos genéricos em si:

```
public class HistoryBox<T> extends Box<T> {
    private ArrayList<T> history = new ArrayList<>();

    public void set( T newObj ) {
        object = newObj;
        history.append( newObj );
    }
}

Box<Pessoa> caixaEspecial = new HistoryBox<>();
```

Herança

Mas cuidado: derivação dos parâmetros não implica derivação do tipo genérico

```
public class Box<T> {
    private T object;

    public void set( T newObj ) { object = newObj; }
    public T get() {
        return object;
    }
}

Box<Aluno> caixaAluno = new Box<>();
Box<Pessoa> caixaPessoa = caixaAluno;
// tipos incompatíveis
```

Wildcards

Tipos especiais, utilizados para generalizar o uso de tipos genéricos como parâmetros de métodos:

```
public static boolean caixaVazia( Box<?> caixa ) {
    return (caixa.get() == null);
}
public static String nomeDentro( Box<? extends Pessoa> caixa ) {
    return caixa.get().getNome(); // TODO: e se get() == null
}
public static void setProfessor( Box<? super Professor> caixa ) {
    caixa.set(new Professor("Jefferson"));
}
```

Comentários adicionais

- Classes e métodos genéricos também podem ser parametrizados por tipos genéricos
  - `ArrayList< ArrayList<Integer> >`
- Métodos e classes genéricos podem depender de múltiplos tipos
  - `<T,U,V,...>`

Exercícios

1. Implemente um método genérico que troque de posição dois elementos de um array (parâmetros: o array e as duas posições)
2. Implemente uma classe genérica que armazene uma quantidade fixo de números de um tipo qualquer (parâmetro do construtor: capacidade). Crie métodos para:
   - 2.1 retornar quantos espaços livres existem;
   - 2.2 armazenar um novo número no primeiro espaço livre (se houver espaço)
   - 2.3 retirar e retornar o número (se houver algum) que está em uma certa posição, sem deixar "buracos"

Dica: As classes padrão de número (como `Integer` e `Float`) são subclasses de uma classe abstrata: `Number`

Créditos

Material baseado na documentação oficial do Java.