INSTITUTO FEDERAL
Rio Grande do Sul

Genéricos

PROF. TIAGO MORAES

Roteiro

- Introdução
- Métodos genéricos
- Tipos limitados
- Classes genéricas
- Erasure
- Coringas (WildCards)

Introdução

- Por que utilizar Genéricos:
  - Criar métodos com lógica repetida (para tipos diferentes) uma vez só:
    - Exemplo: algoritmo de ordenação de vetores (a lógica é independente do tipo)
  - Criar uma classe parametrizada por um tipo, ao invés de criar várias classes semelhantes
    - Exemplo: listas
  - Possibilitar a segurança de tipos em tempo de compilação (**type safity**)
    - Elimina a necessidade de Casting
      - Antes dos genéricos, se utilizava o Object e precisava-se realizar *castings* (o que pode gerar erros de execução)

Introdução

- Nomemclatura:
  - genéricos (*generics*)
  - programação genérica
  - polimorfismo paramétrico
  - tipos parametrizados ou tipo genérico (classe ou interface parametrizada)

- Introduzida no J2SE5

- Permite criar **métodos**, **classes** e **interfaces genéricos**
  - onde o tipo é passado por parâmetro
  - isola a lógica, que eventualmente se repete para diferentes tipos

Métodos Genéricos

- Definição do tipo por parâmetro
  - Se define o tipo antes do retorno de um método com o uso de conchetes angulares “<>”

```
public <T> void metodo(){ ...}
```

  - Após parametrizado, o método pode: retornar, receber parâmetros e ter variáveis locais do tipo parametrizado

```
public <T> T metodo(T par1, T par2){
    T varLocal;
}
```

  - Boa prática: utilizar uma letra maiúscula: T, E etc

Métodos Genéricos

- Exemplo:

```
public class Teste{
    public static <T> void printa(T param){
        System.out.println(param);
    }
}
```

Uso do método genérico

```
public static void main(String[] args){
    Teste.printa("ola");
    Teste.printa(52);
}
```

- O compilador busca **um, e somente um**, método que corresponda a chamada, senão ocorrerá um erro de compilação.

Métodos Genéricos

- Pode-se passar mais de um tipo por parâmetro (nos colchetes angulares), basta definir outro nome.
- Exemplo:

```
public class Teste{
    public <T, V> boolean mesmoTipo(T p1, V p1){
        return p1.getClass().equals(p2.getClass());
    }
}
```

Uso do método genérico

```
public static void main(String[] args){
    Teste t1 = new Teste();
    System.out.println(t1.mesmoTipo("5",5)); //printa false

    System.out.println(t1.mesmoTipo(8,5)); //printa true
}
```

Tipos limitados

- Tipos limitados:
  - Pode-se limitar os possíveis tipos que podem ser passados por parâmetro
  - Define-se que o parâmetro tem que ser um “subtipo” de uma classe ou interface
    - Utiliza-se a palavra chave “extends” (mesmo para interfaces)

- Exemplo: para o parâmetro ser um número:
  - Number é uma classe Abstrata, superclasse de Integer e Double por exemplo

```
<T extends Number>
```

- Exemplo: para o parâmetro ser um Comparable:
  - Comparable é uma interface que garante que o compareTo está implementado

```
<T extends Comparable<T>>
```

Tipos Limitados

- Tipos limitados – Exemplo:
  - Método genérico que retorna o inverso de um número:

```
public class Teste{
    public <T extends Number> double inverso(T p1){
        return 1/p1.doubleValue();
    }
}
```

Classes Genéricas

- Classes também podem ser parametrizadas, esses parâmetros de tipo poderão ser utilizados em todo escopo da classe:
  - Atributos
  - Métodos

```
public NomeClasse<T> { ...}
```

- As classes também podem ter mais de um parâmetro
- Os parâmetros também podem ser limitados

Classes Genéricas

- Exemplo:

```
public class Pacote <T>{
    private T conteudo;
    public T getConteudo(){
        return this.conteudo;
    }
    public void setConteudo(T param){
        this.conteudo = param;
    }
}
```

Uso da classe genérica

```
public static void main(String[] args){
    Pacote<String> pac1 = new Pacote<>();
    pac1.setConteudo("ola");

    Pacote<Double> pac2 = new Pacote<>();
    pac2.setConteudo(58.6);
    //pac2.setConteudo("oi");      //erro de compilação

}
```

Classes Genéricas

- Antes dos genéricos, programava-se genericamente com o uso da classe Objetc, porém isso acarreta o problema dos castings
  - Exemplo:

```
public class Pacote{
    private Object conteudo;
    public Object getConteudo(){
        return this.conteudo;
    }
    public void setConteudo(Object param){
        this.conteudo = param;
    }
}
```

```
public static void main(String[] args){
    Pacote pac = new Pacote();
    pac.setConteudo("ola");

    //pac.setConteudo(58.6);
    String var1 = (String) pac.getConteudo() //possivel erro de execução
}
```

Erasure

- Processo de tradução em tempo de compilação
- O compilador trata os genéricos da seguinte maneira:
  - Substitui a seção de parâmetro de tipo por tipos reais (Object se nao limitado).
  - E os castings necessários são acrescentados.
    - public <T> void metodo(T par) → public void metodo(Object par)
    - public <T extends Number> void metodo(T par) → public void metodo(Number par)
  - Por isso que a parametrização não pode ser utilizada com tipos básicos:
    - int, float, double
    - Apenas tipos de referencia (classes)
  - Para esse tipo de situação o Java possui as classes Integer, Float, Double etc...

Coringas - wildcards

- Ao se utilizar os genéricos não se pode considerar que uma instância é subtipo de outra pois os seus parâmetros são subtipos

- Por exemplo:
  - Mesmo que Number **seja supertipo** de Double
  - Pacote<Number> **não é supertipo** de Pacote<Double>
    - O supertipo de Pacote<Double> é Object

- Por isso certas construções não são possíveis:

```
static void printa(Pacote<Object> arg){
    System.out.println(arg.getConteudo());
}
public static void main(String[] args){
    Pacote<String> pac1 = new Pacote<>();
    pac1.setConteudo("ola");
    printa(pac1);          //erro
}
```

Coringas - wildcards

- Como é possível então generalizar, já que não se sabe o tipo de Pacote?

```
static void printa(Pacote<Object> arg){
}
```

  - Com o uso do coringa (*wildcard*) "?"

- Exemplo

```
static void printa(Pacote<?> arg){
    System.out.println(arg.getConteudo());
}
public static void main(String[] args){
    Pacote<String> pac1 = new Pacote<>();
    pac1.setConteudo("ola");
    printa(pac1);          //erro
}
```

- O coringa pode ser também limitado:
  - Por exemplo: qualquer Pacote parametrizado com um subtipo de Number:

```
Pacote<? extends Number>
```