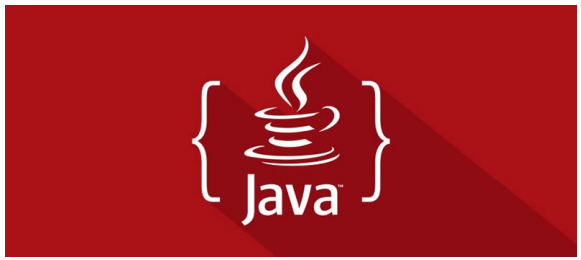
Programação II

Aula 01

Evandro J.R. Silva[1]

[1] Bacharelado em Ciência da Computação
Estácio Teresina



Sumário

Introdução

- *Java Generics* é uma *feature* do Java que permite a criação de estruturas de dados e métodos que não dependem de um tipo específico de dados.
- É bastante utilizado em conjunto com o *Java Collections*.
- Para percebermos uma vantagem óbvia, vejamos alguns exemplos de métodos sobrecarregados (`Sobrecarregados.java`).
- Depois vejamos um método genérico (`Generico.java`).

Métodos Genéricos

- Como vimos no exemplo anterior, um tipo genérico pode ser "descrito" como **T** (letra utilizada como padrão de boas práticas, significando *Type*, ou *Tipo*).
  - Outras letras podem ser utilizadas também. É bastante comum a letra **E**, para elemento, **N**, para número, S, G, etc.
  - Em comum, a declaração do tipo genérico é uma letra maiúscula.
- O Java vai inferir durante a compilação qual o tipo que está sendo utilizado na chamada do método. Após inferir, o tipo genérico será trocado pelo seu tipo verdadeiro.
  - Isso é chamado de ***Type Erasure***,
- Mesmo declarando um tipo genérico, você pode querer limitar os tipos possíveis
  - Isso é chamado de **Tipo Limitado** ou *Bounded Type*. Vejamos um exemplo: `TipoLimitado.java`
  - O Java permite múltiplos limites `<T extends A & B & C>`

Classes Genéricas

- É possível criar classes genéricas também.
- A criação de uma classe genérica é quase idêntica à criação de uma classe comum. A diferença é que na declaração da classe genérica, é necessário declara o tipo genérico (<T>).
- Essa *feature* é interessante de ser utilizada na adaptação/criação de alguma estrutura ou coleção de dados. Vamos ao exemplo: `Pilha.java`

Annotations

- *Annotations* (anotações em tradução livre) são uma forma de **metadados**.
- Proveem dados sobre o programa, sem ser parte do programa, ou seja, não possuem efeito direto.

Annotations

- *Annotations* (anotações em tradução livre) são uma forma de **metadados**.
- Proveem dados sobre o programa, sem ser parte do programa, ou seja, não possuem efeito direto.
- Alguns de seus principais usos:
  - **Informação para o compilador**: anotações podem ser usadas pelo compilador para detectar erros ou suprimir avisos (*warnings*).
  - **Processamento em tempo de compilação e em tempo de implantação**: algumas ferramentas de software podem processar as informações das anotações para gerar código, arquivos XML, etc.
  - **Processamento em tempo de execução**: algumas anotações podem ser examinadas em tempo de execução.

Básico

- Formato

```
@Entidade
```

Básico

- Formato

```
@Entidade
```

- O `@` indica ao compilador que se trata de uma anotação. Vejamos o exemplo de uma anotação que já vimos algumas vezes:

```
@Override
void superMetodo(){ ... }
```

Básico

- Formato

```
@Entidade
```

- O `@` indica ao compilador que se trata de uma anotação. Vejamos o exemplo de uma anotação que já vimos algumas vezes:

```
@Override
void superMetodo(){ ... }
```

- A anotação pode incluir elementos que podem ser nomeados ou não nomeados, e os elementos possuem valores:

```
@Author(
    name = "Prof. Evandro",
    date = "01/09/2002"
)
class Exemplo { ... }
```

OU

```
@SuppressWarnings(value = "unchecked")
void meuMetodo() { ... }
```

Básico

- Se há somente um elemento `value`, ele pode ser omitido:

```
@SuppressWarnings("unchecked")
void meuMetodo() { ... }
```

Básico

- Se há somente um elemento `value`, ele pode ser omitido:

```
@SuppressWarnings("unchecked")
void meuMetodo() { ... }
```

- Se a anotação não tem elementos, os parênteses podems ser omitidos: `@Override`.

Básico

- Se há somente um elemento `value`, ele pode ser omitido:

```
@SuppressWarnings("unchecked")
void meuMetodo() { ... }
```

- Se a anotação não tem elementos, os parênteses podems ser omitidos: `@Override`.
- Múltiplas anotações podem ser usadas na mesma declaração

```
@Author(name = "Prof. Evandro")
@EBook
class Exemplo { ... }
```

Básico

- Se há somente um elemento `value`, ele pode ser omitido:

```
@SuppressWarnings("unchecked")
void meuMetodo() { ... }
```

- Se a anotação não tem elementos, os parênteses podems ser omitidos: `@Override`.
- Múltiplas anotações podem ser usadas na mesma declaração

```
@Author(name = "Prof. Evandro")
@EBook
class Exemplo { ... }
```

- Anotações do mesmo tipo podem ser usadas (*repeating annotation*)

```
@Author(name = "Prof. Evandro")
@Author(name = "Aluno Monitor")
void meuMetodo() { ... }
```

Básico

- Se há somente um elemento `value`, ele pode ser omitido:

```
@SuppressWarnings("unchecked")
void meuMetodo() { ... }
```

- Se a anotação não tem elementos, os parênteses podems ser omitidos: `@Override`.
- Múltiplas anotações podem ser usadas na mesma declaração

```
@Author(name = "Prof. Evandro")
@EBook
class Exemplo { ... }
```

- Anotações do mesmo tipo podem ser usadas (*repeating annotation*)

```
@Author(name = "Prof. Evandro")
@Author(name = "Aluno Monitor")
void meuMetodo() { ... }
```

- Lista das anotações predefinidas pelo Java, e anotações para anotações: java.lang

Básico

- Onde as anotações podem ser utilizadas:
  - Expressão de criação de instância de classe:

```
new @Interned MeuBejeto();
```

  - *Cast* de tipo:

```
estringue = (@NonNull String) str;
```

  - Cláusula `implements`:

```
class ListaImutavel<T> implements @Readonly List<@Readonly T> { ... }
```

  - Declaração de *lançamento* de exceção:

```
void monitorarTemperatura() throws @Critical TemperatureException { ... }
```

Criando uma Annotation

- Imagine-se trabalhando em uma equipe de desenvolvimento.
- E que faz parte da conduta da empresa que você escreva os seguintes dados, sempre que criar uma nova classe:

```
public class QueBonitaASuaRoupa
{
// Autor: Fulanno
// Data: 31/02/2023
// Versao: 140
// Data da ultima versao: 07/03/2023
// Por: Beltrano
// Revisores: Cicrano, Maliciano, Derpina

// codigo da classe ...
}
```

Criando uma Annotation

- Para transformar aqueles comentários em uma annotation (metadata), basta seguir a seguinte sintaxe:

```
@interface DadosDaClasse
{
    String autor();
    String data();
    int versao() default 1;
    String dataUltimaVersao() default "N/A";
    String ultimoMexedor() default "N/A";
    String[] revisores();
}
```

- Se for acrescentado `@Documented` antes da primeira linha, a sua anotação passa a fazer parte da documentação Javadoc.

Criando uma Annotation

- Agora dá pra criar a mesma classe usando a anotação que você criou!

```
@DadosDaClasse(
    autor = "Fulano",
    data = "31/02/2023",
    versao = 140,
    dataUltimaVersao = "07/03/2023",
    ultimoMexedor = "Beltrano",
    revisores = {"Cicrano", "Maliciano", "Derpina"}
)
public class QueBonitaASuaRoupa { ... }
```

Criando uma Annotation

- Agora dá pra criar a mesma classe usando a anotação que você criou!

```
@DadosDaClasse(
    autor = "Fulano",
    data = "31/02/2023",
    versao = 140,
    dataUltimaVersao = "07/03/2023",
    ultimoMexedor = "Beltrano",
    revisores = {"Cicrano", "Maliciano", "Derpina"}
)
public class QueBonitaASuaRoupa { ... }
```

- Vamos ver os exemplos
  `DadosDaClasse.java`, `IsExemplo.java` e `ExemploAnotacao.java`

Exercícios

1. Crie um método genérico que receba dois parâmetros, e então retorne *verdadeiro* se forem iguais e *falso* se forem diferentes.
2. Crie um método genérico que receba um *array* e troque dois elementos de posição.
3. Crie uma classe genérica que simule uma árvore binária, com um limite de profundidade.
4. Crie pelo menos duas anotações e atualize os códigos das classes que você já criou.

FIM

- Levando em consideração que vocês já sabem Java, essa aula foi apenas uma pequena revisão do Java.
- **Tarefa de casa**: Capítulos 2, 3, 4 e 5 do livro **Java: Como Programar** do Deitel, oitava edição. Disponível nas bibliotecas virtuais (creio que na bilbioteca física também).
- Os capítulos citados são apenas para revisar o Java!

Até a próxima!