Programação II

Aula 04

Evandro J.R. Silva[1]

[1] Bacharelado em Ciência da Computação
Estácio Teresina

01/09/2022

Sumário

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 UnmodifiableList<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.
- 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 { ... }
```

Exercícios

Começando pelos métodos e classes genéricas:

1. Escreva um método genérico para trocar a posição de dois elementos em um array.
2. Escreva um método genérico para encontrar o maior elemento na faixa [começo, fim)* de uma lista. (*colchete $\rightarrow$ incluso; parêntese $\rightarrow$ não incluso).
3. Escreva uma classe genérica que funcione como uma coleção de objetos genéricos. Esta classe deverá ter um método genérico para contar o número de elementos em uma coleção que tenha uma propriedade específica (por exemplo, números ímpares).

Agora crie anotações e atualize os códigos anteriores com as anotações criadas.

FIM

Só conseguimos aprender a programar quando treinamos!

Até a próxima!