Java Avançado

Annotations e a Reflection API

SoftBlue
www.softblue.com.br

Todos os direitos de cópia reservados. Não é permitida a distribuição física ou eletrônica deste material sem a permissão expressa e por escrito do autor.

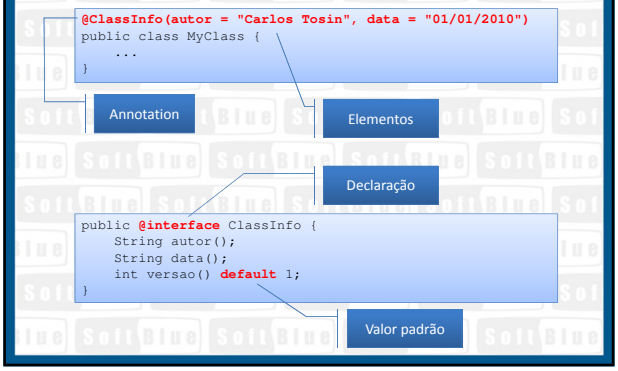
Tópicos Abordados

- Annotations
  - O que são
  - Como usar e criar
  - Elemento *Value*, *@Retention*, *@Target*
  - *@Override* e *@SuppressWarnings*
- Reflection API
  - Objeto *Class*
  - Instanciando objetos
  - Invocando métodos

O Que São Annotations

- Mecanismo criado a partir do Java 5
- São declarações no código que podem ser usadas por ferramentas ou programas externos
- Não influenciam diretamente na execução do código

Usando e Criando Annotations



O Elemento Value

- Quando a annotation possui apenas um elemento **value**, ele pode ser omitido quando a anotação é usada

```
public @interface Autor {
    String value();
}
```

```
@Autor("Carlos Tosin")
public class MyClass() {
    ...
}
```

@Retention

- Configura como a anotação deve se comportar perante ao compilador e à JVM
- Tipos
  - RetentionPolicy.RUNTIME
    - A anotação pode ser lida em tempo de execução pela JVM
  - RetentionPolicy.CLASS
    - A anotação é lida pelo compilador mas não pode ser lida em tempo de execução
  - RetentionPolicy.SOURCE
    - A anotação é ignorada pelo compilador

Exemplo de @Retention

```
@Retention(RetentionPolicy.RUNTIME)
public @interface ClassInfo {
    String autor();
    String data();
    int versao() default 1;
}
```

@Target

- Indica em qual elemento a annotation pode ser aplicada
- Tipos
  - ElementType.TYPE
    - Classe, interface ou enum
  - ElementType.METHOD
    - Métodos
  - ElementType.FIELD
    - Atributos
  - etc.

Exemplo de @Target

```
@Target(ElementType.TYPE)
public @interface ClassInfo {
    String autor();
    String data();
    int versao() default 1;
}
```

A Anotação *@Override*

- Indica que um método sobrescreve outro
- É opcional, mas quando utilizada gera erro de compilação se o método anotado não estiver sobrescrevendo um método da superclasse

```
@Override
public String toString() {
  return "...";
}
```

Se o método **toString()** não existir na superclasse, gera erro de compilação

A Anotação *@SuppressWarnings*

- Utilizada para remover mensagens de *warning* do código
- O seu uso mais comum é para remover mensagens de conversão de tipos quando o generics é utilizado
- Pode anotar classes, métodos e código

A Anotação *@SuppressWarnings*

```
List<String> l = new ArrayList();
```

Warning: **The expression of type ArrayList needs unchecked conversion to conform to List<String>**

```
@SuppressWarnings("unchecked")
List<String> l = new ArrayList();
```

O *warning* desaparece

A Reflection API

- A Reflection API do Java permite que as classes conheçam sobre suas estruturas internas
- O ponto de entrada é um objeto **Class**, que guarda informações sobre uma classe

```
Class c = String.class;
```

```
Class c = Class.forName("java.lang.String");
```

```
Class<String> c = String.class;
```

```
Class<String> c = (Class<String>) Class.forName("java.lang.String");
```

O Objeto Class

- O objeto Class pode representar também outros elementos que não sejam classes, como interfaces e enums
- A partir do objeto Class, é possível descobrir quais são os atributos, construtores e métodos

Método	Descrição
getFields()	Retorna um array de atributos
getField()	Retorna um atributo específico
getConstructors()	Retorna um array de construtores
getConstructor()	Retorna um construtor específico
getMethods()	Retorna um array de métodos
getMethod()	Retorna um método específico

Instanciando Objetos

- Com a Reflection API, é possível instanciarmos objetos quando conhecemos apenas o nome da classe

```
Class c = Class.forName("br.com.softblue.MyClass");
MyClass m = (MyClass) c.newInstance();
```

Invocando Métodos

- Outro uso comum da Reflection API é para invocar métodos

```
Class c = Class.forName("br.com.softblue.MyClass");
MyClass o = (MyClass) c.newInstance();

Method m = c.getMethod("imprimir", String.class);
m.invoke(o, "algum texto");
```

Procura o método *imprimir* da classe, que recebe uma String como parâmetro

Invoca o método *m* no objeto *o*, passando o parâmetro para o método

Colocando em Prática...

Agora que você já aprendeu a teoria, acesse as vídeo-aulas práticas e pratique os assuntos abordados neste módulo!

Clique aqui para acessar as vídeo-aulas práticas