Аннотации

# Введение

Аннотации представляют собой некую мета-информацию. Они не выполняют какого-либо действия сами по себе, но они могут предоставлять дополнительную информацию, которая может быть использована компилятором, различными утилитами сборки и генерации кода, а также они могут обрабатываться во время выполнения программы.

Способы использования аннотаций:

* **Информация для компилятора** — аннотации могут быть использованы компилятором для определения ошибок или подавления предупреждений.
* **Обработка времени компиляции/сборки/развертывания** — средства разработки программного обеспечения могут обрабатывать аннотации для генерации кода, xml-файлов и т.д.
* **Обработка времени исполнения** — использование аннотаций во время исполнения (наиболее интересующий нас кейс работы с ними).

# Обзор синтаксиса аннотаций

Аннотации предваряются знаком собачки. Пример часто используемой аннотации @Override, которая указывает компилятору, что этот метод переопределяет базовый метод:

@Override  
**public void** someMethod1() {  
 *// ...*}

Аннотации могут иметь элементы:

@SuppressWarnings(value = **"unchecked"**)  
**public void** method1() {  
}

Если элементов много, то они разделяются запятой, если элемент только один, и его имя value , то его название можно не указывать:

@SuppressWarnings(**"unchecked"**)  
**void** myMethod() { ... }

# Предопределённые аннотации

## Предопределённые аннотации из пакета java.lang

*@Deprecated*  указывает, что элемент устарел и не должен использоваться. Компилятор Java генерирует предупреждение, если вы используете класс, метод или поле, помеченные аннотацией *@Deprecated*. Устаревший элемент должен быть также помечен тегом JavaDoc *@deprecated* :

*/\*\*  
 \** ***@deprecated*** *\* explanation of why it was deprecated  
 \*/*@Deprecated  
**static void** deprecatedMethod() { }

*@Override*  указывает, что метод переопределяет метод базового класса. Эту аннотацию использовать не обязательно, но рекомендуется.

@Override  
**public int** methodName(**double** x) {}

Если метод с *@Override*  не может корректно переопределить метод базового класса, то компилятор генерирует ошибку.

*@SuppressWarnings* подавляет предупреждения. Пример подавления предупреждения об устаревшем методе:

*// Указываем компилятору не генерировать  
// предупреждение об использовании  
// устаревшего метода.*@SuppressWarnings(**"deprecation"**)  
**void** useDeprecatedMethod() {  
 *// Используем устаревший метод.* objectOne.deprecatedMethod();  
}

Каждое предупреждение принадлежит какой-либо категории. В спецификации Java описано две категории. В *@SupressWarnings*  можно указывать несколько категорий:

@SuppressWarnings({**"unchecked"**, **"deprecation"**})

Различные реализации компиляторов и различные IDE могут добавлять свои категории предупреждений. Неподдерживаемые названия категорий пропускаются при использовании*@SuppressWarnings*.

*@SafeVarargs* применяется к методу или конструктору и указывает, что код не осуществляет потенциально опасных операций со своим varargs-параметром (параметр, принимающий произвольное число параметров).

*@FunctionalInterface* указывает, что это объявление типа будет функциональным интерфейсом Java 8.

# Объявление аннотаций

Предположим, что вы по традиции при объявлении каждого нового класса монстра записываете в комментариях информацию об авторах в таком виде:

**class** Goblin {  
 *// author : John Clark  
 // sprites : Izabella Simpson  
 // sound : Michael Lermontov  
 // code : Pushkin A.   
 // createdAt : 2016-03-30  
 // description : Goblin is lurking creature.  
 // comments :   
  
 // ... code*}

Вы можете записывать эту информацию с помощью аннотаций. Для этого вам сперва нужно объявить эту аннотацию:

@**interface** Monstr {  
 String author();  
 String sprites();  
 String sound();  
 String code();  
 String createdAt() **default "0000-00-00"**;  
 String description();  
  
 *// Example of array use* String[] comments();  
}

Теперь можно применить его к нашему классу Goblin:

@Monstr(  
 author = **"John Clark"**,  
 sprites = **"Izabella Simpson"**,  
 sound = **"Michael Lermontov"**,  
 code = **"Pushkin A."**,  
 createdAt = **"2016-03-30"**,  
 description = **"Goblin is lurking creature"**,  
 comments = {**"lol"**, **"gg"**, **"Аффтар жжёт"**}  
)  
**class** Goblin {  
}

Обратите внимание на запись элементов аннотации ( author ,sprites , sound ) и на запись массива элементов с использованием фигурных скобок ( comments ).

Можно использовать предопределённую аннотацию *@Documented*, чтобы наша аннотация попадала в документацию, сгенерированную утилитой JavaDoc:

**import** java.lang.annotation.Documented;  
  
@Documented  
@**interface** Monstr {  
 String author();  
 String sprites();  
 String sound();  
 String code();  
 String createdAt() **default "0000-00-00"**;  
 String description();  
  
 *// Example of array use* String[] comments();  
}

Атрибуты могут иметь только следующие типы:

* примитивы
* String
* Class или «any parameterized invocation of Class»
* enum
* annotation
* массив элементов любого из вышеперечисленных типов

# Мета-аннотации

Аннотации, применяемые к другим аннотациям, называются **мета-аннотациями**. Есть несколько мета-аннотаций в пакетеjava.lang.annotation :

*@Retention* определяет, как аннотация будет сохранена:

RetentionPolicy.SOURCE — аннотация будет только в исходном коде, и она будет игнорироваться компилятором.

RetentionPolicy.CLASS — аннотация будет доступна компилятору, но но будет игнорироваться виртуальной машиной Java.

RetentionPolicy.RUNTIME — аннотация будет сохраняться JVM и будет доступна во время выполнения.

*@Target* — указывает какие элементы можно помечать этой аннотацией:

ElementType.ANNOTATION\_TYPE — данная аннотация может быть применена к другой аннотации.

ElementType.CONSTRUCTOR — может быть применена к конструктору.

ElementType.FIELD — может быть применена к полю.

ElementType.LOCAL\_VARIABLE — может быть применена к локальной переменной.

ElementType.METHOD — может быть применена к методу.

ElementType.PACKAGE — может быть применена к пакету.

ElementType.PARAMETER — может быть применена к параметрам метода.

ElementType.TYPE — может быть применена к любому элементу класса.

*@Documented* — указывает, что элементы, помеченные этой аннотацией, должны документироваться JavaDoc. По умолчанию аннотации не включаются в документацию.

*@Inherited* — аннотация может быть унаследована от базового класса (по умолчанию не наследуются). Когда запрашивается аннотация класса, и у класса нет такой аннотации, то запрашивается аннотация базового класса. Эта аннотация может быть применена только к классам.

*@Repeatable* — аннотация может быть применена несколько раз.

Допустим, мы хотим применить аннотацию *@Author*  несколько раз для указания нескольких авторов:

@Author(**"Petya"**)  
@Author(**"Vasya"**)  
@Author(**"Suslik"**)  
**class** Goblin {  
}

Тогда мы должны объявить такую аннотацию вот так:

**import** java.lang.annotation.Repeatable;  
  
@Repeatable(Authors.**class**)  
@**interface** Author {  
 String value();  
}

Обратите внимание, что добавлена аннотация *@Repeatable*  с указанием Authors.class , который мы должны объявить как аннотацию с массивом аннотация Author:

@**interface** Authors{  
 Author[] value();  
}

Теперь мы можем указывать аннотацию *@Author*  столько раз, сколько захотим, для любого класса.

# Обработка аннотаций в RUNTIME:

По умолчанию, у всех аннотаций стоит RetentionPolicy.CLASS. Это мне кажется недодумкой. В исходниках JDK очень часто используется эта policy, но вот в разработке нужна именно RetentionPolicy.RUNTIME.

Посредством рефлексии получается необходимый вам класс Class/Method/Field, у которого имеются следующие методы для работы с аннотациями:

Boolean [**isAnnotationPresent**](https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/AnnotatedElement.html#isAnnotationPresent-java.lang.Class-)([**Class**](https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html)<? extends [**Annotation**](https://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Annotation.html)> annotationClass)

Returns true if an annotation for the specified type is *present* on this element, else false.

[**Annotation**](https://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Annotation.html)[] [**getAnnotations**](https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/AnnotatedElement.html#getAnnotations--)() -

Returns annotations that are *present* on this element.

[**Annotation**](https://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Annotation.html)[] [**getDeclaredAnnotations**](https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/AnnotatedElement.html#getDeclaredAnnotations--)()

Returns annotations that are *directly present* on this element.

<T extends [**Annotation**](https://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Annotation.html)> T [**getAnnotation**](https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/AnnotatedElement.html#getAnnotation-java.lang.Class-)([**Class**](https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html)<T> annotationClass) -

Returns this element's annotation for the specified type if such an annotation is *present*, else null.

<T extends [**Annotation**](https://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Annotation.html)> T[] [**getAnnotationsByType**](https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/AnnotatedElement.html#getAnnotationsByType-java.lang.Class-)([**Class**](https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html)<T> annotationClass)

Returns annotations that are *associated* with this element.

<T extends [**Annotation**](https://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Annotation.html)> T [**getDeclaredAnnotation**](https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/AnnotatedElement.html#getDeclaredAnnotation-java.lang.Class-)([**Class**](https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html)<T> annotationClass)

Returns this element's annotation for the specified type if such an annotation is *directly present*, else null.

<T extends [**Annotation**](https://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Annotation.html)> T[] [**getDeclaredAnnotationsByType**](https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/AnnotatedElement.html#getDeclaredAnnotationsByType-java.lang.Class-)([**Class**](https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html)<T> annotationClass)

Returns this element's annotation(s) for the specified type if such annotations are either *directly present* or *indirectly present*.