Аннотации

**Аннотация** — это специальная конструкция языка, связанная с классом, методом или переменной, предоставляющая программе дополнительную информацию, на основе которой программа может предпринять дальнейшие действия или реализовать дополнительную функциональность, такую как генерация кода, проверка ошибок и т. д.

Аннотации начинаются с символа **@.**

Стандартные аннотации

**@Override** - используется для обозначения переопределенного метода из базового класса.

**@SuppressWarnings -** используется для подавления предупреждений компилятора.

**@Deprecated -** используется для пометки устаревших методов или типов.

**@FunctionalInterface -** используется для указания того, что в интерфейсе не может быть более одного абстрактного метода.

**@SafeVarargs -** позволяет подавить предупреждения выдающеюся при использовании в аргументах метода обобщенных типов.

**@Retention** - задает "тип хранения" аннотации над которой она указана.

**@Target** - задает тип объекта над которым может указываться создаваемая нами аннотация.

Пользовательские аннотации

**Мы можем создавать свои аннотации, например, для реализации следующей функциональности:**

* Уменьшение дублирования кода.
* Автоматизация генерации бойлерплейт кода.
* Отлов ошибок во время компиляции, например, потенциальные Null Pointer Exception.
* Настройка поведения в рантайме на основе наличия аннотации.

@Company{

name="ABC"

city="XYZ"

}

**public** **class** **CustomAnnotatedEmployee** {

...

}

При создании экземпляров класса CustomAnnotatedEmployee все экземпляры будут содержать одно и то же название компании (name) и города (city) — больше не нужно добавлять эту информацию в конструктор.

Создать пользовательскую аннотацию можно с помощью ключевого слова @interface:

**public** @interface Company{}

Мета-аннотации

**Мета-аннотации** — это аннотации, применяемые к другим аннотациям для предоставления информации об аннотации компилятору или среде выполнения.

**Мета-аннотации могут ответить на следующие вопросы об аннотации:**

1. Может ли аннотация наследоваться дочерними классами?
2. Должна ли аннотация отображаться в документации?
3. Можно ли применить аннотацию несколько раз к одному и тому же элементу?
4. К какому типу элементов можно применить аннотацию: к классу, методу, полю и т.д.?
5. Обрабатывается ли аннотация во время компиляции или в рантайме?

**@Inherited** - по умолчанию аннотация не наследуется от родительского класса к дочернему. Мета-аннотация позволяет ей наследоваться.

**@Documented** - указывает, что аннотация должна присутствовать в JavaDoc.

**@Repeatable** - позволяет использовать аннотацию несколько раз на одном методе, классе или поле. Для использования @Repeatable — аннотации необходимо создать аннотацию-контейнер, которая хранит значение в виде массива исходных аннотаций.

**@Target** - определяет типы элементов, к которым может применяться аннотация.

**Существуют следующие типы целей, названия которых говорят сами за себя:**

* ElementType.ANNOTATION\_TYPE
* ElementType.CONSTRUCTOR
* ElementType.FIELD
* ElementType.LOCAL\_VARIABLE
* ElementType.METHOD
* ElementType.PACKAGE
* ElementType.PARAMETER
* ElementType.TYPE

**@Retention -** указывает, когда аннотация будет доступна:

* SOURCE — аннотация доступна в исходном коде и удаляется после компиляции.
* CLASS — аннотация сохраняется в class-файле во время компиляции, но недоступна при выполнении программы.
* RUNTIME — аннотация доступна в рантайме.

Oracle рекомендует писать тэги в следующем порядке:

**@author** — документирует автора кода;

**@version** — обеспечивает только одно обновление за раз (позволяет избежать блокировки);

**@param** — документирует имя и описание параметров;

**@return** — документирует возвращаемое значение; опускать, если возвращает пустоту (void);

**@throws** — документирует проверенные исключения (объявленные в throws);

**@see** — ссылка или указание на ссылку;

**@since** — документирует версию продукта, с которой была добавлена новая функциональность;

**@deprecated** — документирует, что код больше не поддерживается.

Классификация аннотаций

Аннотации можно классифицировать по количеству передаваемых в них параметров: без параметров, с одним параметром и с несколькими параметрами.

**Маркерные аннотации**

Маркерные аннотации не содержат никаких членов или данных. Для определения наличия аннотации можно использовать метод isAnnotationPresent().

**Аннотации с одним значением**

Аннотации с одним значением содержат только один атрибут, который принято называть value.

**Полные аннотации**

Они состоят из нескольких пар "имя-значение".

Тест-аннотации

**@Test** **-** сообщает JUnit, что аннотированный метод должен выполняться как тест. Чтобы запустить метод, JUnit создает новый экземпляр класса, а затем вызывает тестовый метод.

**Для этой аннотации можно указать два необязательных параметра:**

**@Timeout -** приводит к сбою метода тестирования, если выполнение занимает больше времени, чем указанное время, измеренное на часах в миллисекундах.

**@Expected -** объявляет, что тестовый метод должен выдавать определенное исключение, в противном случае тест завершится неудачей.

**@Ignore -** указывает игнорировать тест или группу тестов, чтобы избежать потенциального сбоя при выполнении.

**@Before** – методы выполняются перед каждым тестом. Это полезно, когда вы хотите выполнить некоторый код перед запуском теста, например, настроить тестовую среду.

**@After** — это противоположность предыдущему тэгу. Все методы, помеченные @After, будут запущены после теста.

**Методы @AfterAll или @AfterClass** выполняются после выполнения всех тестов класса. Все методы, аннотированные @beforeAll и @afterAll должны быть статическими, так как выполняются перед запуском тестов класса.

Аннотация @Documented

По умолчанию аннотации не включаются в javadoc. Аннотация, помеченная **@Documented** информирует, что такая аннотация должна быть задокументирована с помощью инструмента javadoc.

Аннотация @Native

Начиная с Java 8, в пакете java.lang.annotation появилась новая аннотация под названием @Native, применимая только к полям. Она указывает, что аннотированное поле является константой, на которую можно ссылаться с нативного кода.