Аннотации

Аннотации

Главной задачей аннотаций является статическое расширение классов (именно классов, а не объектов), путём добавления метаданных в класс, без изменения его методов и свойств.

- **Информация для компилятора** - аннотации могут быть использованы компилятором для обнаружения ошибок (например, @Override) или для подавления предупреждений (@SuppressWarnings);
- **Дополнительная обработка кода во время компиляции** - внешнее программное обеспечение может использовать аннотации для генерации кода или разного рода файлов конфигураций;
- **Дополнительная функциональность кода за счет обработки аннотаций во время выполнения программы** - некоторые аннотации доступны во время выполнения программы.

```java
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import java.lang.annotation.ElementType;

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Mammal {
      String sound();
      int color();
}
```

Ограничения

Несколько ограничений, накладываемых на аннотации:

- объявляемый метод не должен иметь параметров;
- объявление метода не должно содержать ключевое слово throws;
- метод должен возвращать одно из следующих типов: любой примитивный тип, String, Class, enum или массив указанных типов;

Использование

Аннотации могут использоваться со следующими элементами программы:

- класс, интерфейс или перечисления (enum);
- свойства (поля) классов;
- методы, конструкторы и параметры методов;
- локальная переменная;
- блок catch;
- пакет (java package);
- другая аннотация.

Умолчания

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Mammal {
      String sound();
      int color() default 0xffffff;
}
```

Использование

```
@Mammal(color = 0xFFA844, sound = "uuuu")
class Giraffe {
}
```

@Target

Аннотацией @Target указывается, какой элемент программы будет использоваться аннотацией.

- **PACKAGE** - назначением является целый пакет (package);
- **TYPE** - класс, интерфейс, enum или другая аннотация:
- **METHOD** - метод класса, но не конструктор (для конструкторов есть отдельный тип CONSTRUCTOR);
- **PARAMETER** - параметр метода;
- **CONSTRUCTOR** - конструктор;
- **FIELD** - поля-свойства класса;
- **LOCAL_VARIABLE** - локальная переменная (данная аннотация не может быть прочитана во время выполнения программы);
- **ANNOTATION_TYPE** - другая аннотация (мета-аннотация).

```
@Target({ElementType.TYPE, ElementType.TYPE})
```

@Retention

Аннотация @Retention позволяет указать, в какой момент жизни программного кода будет доступна аннотация (java.lang.annotation.RetentionPolicy):

- **SOURCE** - аннотация доступна только в исходном коде и сбрасывается во время создания .class файла;
- **CLASS** - аннотация хранится в .class файле, но недоступна во время выполнения программы;
- **RUNTIME** - аннотация хранится в .class файле и доступна во время выполнения программы.

Пример

```
class User {
    public static enum Permission {
        USER_MANAGEMENT, CONTENT_MANAGEMENT
    }

    private List<Permission> permissions;

    public List<Permission> getPermissions() {
        return new ArrayList<Permission>(permissions);
    }

    // ...
}

@Retention(RetentionPolicy.RUNTIME)
public @interface PermissionRequired {
    User.Permission value();
}
```

Пример

```
@Retention(RetentionPolicy.RUNTIME)
public @interface PermissionRequired {
    User.Permission value();
}
```

Пример

```
@PermissionRequired(User.Permission.USER_MANAGEMENT)
class UserDeleteAction {
    public void invoke(User user) { /* */ }
}

User user = ...;
Class<?> actionClass = ...;
PermissionRequired permissionRequired =
      actionClass.getAnnotation(PermissionRequired.class);
if (permissionRequired != null)
    if (user != null && user.getPermissions().contains(
      permissionRequired.value())) {
        // выполнить действие
    }
```

Пример-2

```
@interface Version {
    int value();
    String author() default "UNKNOWN";
}

@interface History {
    Version[] value() default {};
}

@History({
    @Version(1),
    @Version(value = 2, author = "Jim Smith")
})
class MyClass {}
```

Пример-3

```
interface Serializer<T> {
    void toStream(T obj, OutputStream out) throws IOException;
}

class MySerializer implements Serializer<MyClass> {
    public void toStream(MyClass obj, OutputStream out)
    throws IOException {
        throw new UnsupportedOperationException();
    }
}

@interface SerializedBy {
    Class<? extends Serializer> value();
}

@SerializedBy(MySerializer.class)
class MyClass {}
```