

ANNOTATIONS AND META-ANNOTATIONS

ANNOTATIONS AND META- ANNOTATIONS



ANNOTATIONS (ЧАСТ 1) - ОСНОВИ



Съдържание

7

- Анотации Java (част 1) – 1ч.
 - ▣ Определение
 - ▣ Употреба
 - ▣ Правила за работа
 - ▣ Дефиниране на тип Анотация (annotation) - начин на употреба
 - ▣ Анотиране на анотации
 - ▣ Упражнение

Тип анотация - дефиниция

8

- *Annotations* предоставя **данни в програмата**, които НЕ СА част от самата програма (т.нар. *мета-данни*);
- Нямат директен ефект върху действието на кода, който аотират;
- **Инструментите**, които четат и интерпретират аотации могат да **предоставят различна функционалност** в зависимост от извлечената информация от мета-данните.

История: спецификация „Java Metadata“

9

- Annotations – са добавени към Java 5.0 като резултат от JSR 175 "A Metadata Facility for the Java™ Programming Language" (J2SE 5.0).
- в JavaEE – връзка с външни ресурси и конфигурационни данни директно в програмен код на Java
- Основните анотации са специфицирани в:
 - ▣ JSR-175, [A Metadata Facility for the Java Programming Language](#), и
 - ▣ JSR-250, [Common Annotations for the Java Platform](#).
- Анотациите, характерни за уеб компонентите са описани в [Java Servlet 2.5 Specification](#) (for example).

Употреба на анотациите

10

1. **Информация за компилатора** - за откриване на грешки или потискане на предупреждения (suppress warnings).
2. **Обработка по време на компилиране** и разгръщане на приложението - **софтуерните инструменти** могат да обработват информацията, налична в анотациите с различна цел, като генериране на код, XML файлове и т.н.
Например: могат да подсиgurят консистентност между класове, да се проверява за валидност на аргументи по време на изпълнение и генериране на голяма част от базовият код в прокта.
3. **Обработка по време на изпълнение** — част от анотациите се оценяват по време на изпълнение (runtime).

Елементи на приложение

11

Анотации на програмен код могат да бъдат приложени към различни **програмни декларации** (т.нар. Java елементи), **по-важните** от които са:

- Класове (classes)
- Полета (fields)
- Методи (methods)

Правила при аотиране (1/3)

12

Правило 1: Аотациите се пишат първи, често (**по конвенция**) на собствен ред и може да съдържа елементи с именовани или неименовани стойности, например:

■ **Пример 1**:

```
@Author(  
    name = "Author Name",  
    date = "04/10/2010"  
)  
class MyClass { }
```

Правила при аотиране (2/3)

13

Правило 2: Ако съществува само един елемент с име value (виж пример 2а), то тогава името value **МОЖЕ** да бъде пропуснато (виж пример 2б).

□ **Пример 2а:**

```
@SuppressWarnings(value = "unchecked")
```

```
void myMethod() { }
```

□ **Пример 2б:**

```
@SuppressWarnings("unchecked")
```

```
void myMethod() { }
```

Правила при аотиране (3/3)

14

Правило 3: Ако аотацията няма елементи (3а), скобите могат да бъдат изпуснати (3б).

□ **Пример 3а:**

@Override()

```
void mySuperMethod() { }
```

□ **Пример 3б:**

@Override

```
void mySuperMethod() { }
```

Пример – коментари в код

Условие: екипите от разработчици започват традиционно
тялото на всеки клас с коментари

Случай на употреба: дефиниране на анотация (за документиране) (1/4)

16

Част от анотациите заменят информацията, която се пише в коментари до момента. Екипите от разработчици започват традиционно тялото на всеки клас с коментари, като долният, предоставящи важна информация:

■ Пример (без анотации):

```
public class Generation3List extends Generation2List {  
    // Author: Milen Petrov  
    // Date: 2013-09-12  
    // Current revision: 2  
    // Last modified: 2013-09-12  
    // By: MP  
    // Reviewers: AA, S1, S2  
    // class code goes here  
}
```

Случай на употреба: Дефиниране на анотация (2/4)

17

Дефиниране на анотация за разглежданият случай, става по следният начин:

- Пример (дефиниране на анотация):

```
@interface ClassHeader {  
    String author();  
    String date();  
    int currentRevision() default 1;  
    String lastModified() default "N/A";  
    String lastModifiedBy() default "N/A";  
    String[] reviewers(); // Note use of array  
}
```

Случай на употреба: Дефиниране на елемент от тип анотация – правила: (3/4)

18

- **annotation type definition** – прилича на дефиницията на елемент *interface*, където ключовата дума *interface* е предхождана от символа @ (@ = "AT" както в Annotation Type). Елементите от тип анотация са вид интерфейси (*interface*).
- Тялото на дефинираната (по-горе) анотация съдържа декларации на елементи от тип анотация (***annotation type element***), които приличат на методи.
- Елементите на анотацията може да съдържат множество опционални стойности по 'подразбиране' (default)

Случай на употреба: Пример: Операция по аотиране (употреба / 'извикване') на аотацията (4/4)

19

След като аотацията е създадена, може да се използва по следният начин::

```
@ClassHeader (  
    author = "Milen Petrov",  
    date = "2013-09-12",  
    currentRevision = 2,  
    lastModified = "2013-09-12 ",  
    lastModifiedBy = "Jane Doe",  
    reviewers = {"AA", "S1", "S2"} // Нотация за масив  
)  
public class Student extends User {  
    // ... програмен код на класът ...  
}
```


Случай на употреба: Анотацията @Documented

20

Забележка: За да се появи информация в документацията за **@ClassHeader** – генерирана чрез **Javadoc**, трябва да анотирате дефиницията на анотацията **@ClassHeader** чрез анотацията **@Documented**:

■ Пример:

```
import java.lang.annotation.*; //за да е достъпна @Documented
```

@Documented

```
@interface ClassHeader {  
    // дефиниране елементите на анотацията  
}
```

Резултат: Вградена анотация @Documented резултат(1/2)

21

All Classes

[ClassHeader](#)

[Student](#)

[Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [REQUIRED](#) | [OPTIONAL](#)

DETAIL: [ELEMENT](#)

Annotation Type ClassHeader

```
@Documented  
public @interface ClassHeader
```

Required Element Summary

<code>java.lang.String</code>	author
<code>java.lang.String</code>	date
<code>java.lang.String</code> <code>[]</code>	reviewers

Резултат: Вградена анотация @Documented резултат(2/2)

22

All Classes ClassHeader Student	Optional Element Summary	
	int	currentRevision
	java.lang.String	lastModified
	java.lang.String	lastModifiedBy
Element Detail		
author		
public abstract java.lang.String author		
date		
public abstract java.lang.String date		
reviewers		
public abstract java.lang.String[] reviewers		
currentRevision		
public abstract int currentRevision		
Default:		
1		

Задачи за изпълнение

23

Задача 1.1. Дефинирайте и използвайте анотацията `ClassHeader`, както е показано по-горе на слайдовете.

ANNOTATIONS (ЧАСТ 2) АНОТАЦИИ ВГРАДЕНИ В КОМПИЛАТОРА

Съдържание

25

Анотации:

- `@Deprecated`
- `@Override`
- `@SuppressWarnings`

Анотации, използвани от компилатора

26

Съществуват следните три типа анотации, които са предефинирани в спецификацията на езика:

- `@Deprecated`
- `@Override`
- `@SuppressWarnings`

Анотация @Deprecated и документиране при препоръка за неупотреба

Анотации: @Deprecated (1 / 5)

28

- Постоянното обновяване на класовете и интерфейсите в дадено приложение може да премине през много промени от типа:
 - ▣ Добавяне на нови методи
 - ▣ Премахване или промяна на съществуващи методи;

Анотации: @Deprecated (2/5)

29

Сценарий: Ако даден клас или интерфейс (или метод във тях) стане ,остарял' (obsolete) и трябва да се предупреди клиентското приложение да не го употребява. Възможни решения:

1. **Начин 1:** Чрез премахването на метод от интерфейс може да бъде опасно за съществуващите приложения, които разчитат на дадената функционалност;
2. **Начин 2:** *Маркиране на метода* като ,забранен' (deprecated); по този начин се указва приложенията да не разчитат на тази функционалност, т.к. може да бъде премахната в бъдеще;

Анотации: `@Deprecated` (3/5)

30

- ❑ **`@Deprecated`**— анотацията маркира аотирианият с нея елемент като ‚забранен‘ за използване и че не трябва да бъде използван;
- ❑ Компилятора генерира предупредително съобщение когато програма се опитва да използва метод, клас или поле аотириано посредством `@Deprecated` анотация.
- ❑ Когато даден елемент е забранен – може да се използва и стандартната анотация от инструмента `javadoc` **`@deprecated`** таг (виж примера по-долу)
- ❑ Използваният символ "`@`" `javadoc` коментарите и анотациите не е случайно, а поради тяхната концептуална свързаност;
- ❑ Разликата между тага `deprecated` (в `javadoc`) започва с малко "d", докато при анотациите – с главно "D".

Анотации: @Deprecated (4/5)

31

// javadoc коментар

/**

* **@deprecated**

* *обяснение защо да не се използва*

*/

@Deprecated

```
public void deprecatedMethod() {  
}
```

@deprecated
срещу
@Deprecated

Анотации: @Deprecated (5/5) – програмен код

32

```
public class DeprecatedMethodClass {
```

```
/**
```

```
 * This method is obsolete, because is bad!
```

```
 * {@link #newMethod()} to achieve the same affect.
```

```
 */
```

```
 @Deprecated
```

```
 public void oldMethod() {  
     // broken code  
 }
```

```
 public void newMethod() {  
     // fixed code  
 }
```

```
}
```

При **предупреждение**
на компилатора - **виж**
препоръките в
документацията

Какъв проблем виждате в следната декларация?

```
public interface ServerFolder {  
    @Deprecated  
    public void monitor();  
    public void monitorFolder();  
}
```

@Deprecated annotation - препоръка

- Когато се ползва @Deprecated анотацията, **добра практика** е да има документация защо не се препоръчва и какво да се използва вместо това, например:

```
/**  
 * @deprecated  
 * use of monitor() is discouraged, use  
 * use monitorFolder() instead.  
 */
```

Анотация @Override - ключови думи срещу анотации

Вътрешни анотации за компилатора: @Override (1/2)

37

- **@Override** — информира компилатора, че метода пренаписва същият метод от родителския клас:

// метода е пренаписан

```
@Override public String toString() {  
    }  
}
```

- Не е задължителна, но предпазва от грешки;
- Ако метод е аотиран с @Override, но няма съответен метод в родителския клас – компилатора ще изведе грешка.

Вътрешни анотации за компилатора: @Override

(2/2)

38

- Каква е разликата между ключова дума и анотация – предимства и недостатъци

@SuppressWarnings – добри практики и предупреждения

Вътрешни анотации за компилатора:

@SuppressWarnings (1/3)

40

- **@SuppressWarnings**— указва на компилатора да подтисне предупреждения, които нормално генерира при компилация; в такъв случай предупрежденията не се извеждат;

// при използване на deprecated извикване да не се

// генерира предупреждение

```
@SuppressWarnings("deprecation") void deprecatedCall() {
```

```
    student.enrollCourseInSusi4("JavaEE7");
```

//извикване на deprecated метод enrollCourseInSusi4 нормално се

// извиква enrollCourseInSusi5 но за прекъснали студенти

// записването става в програма, а не в курс

```
}
```

Вътрешни анотации за компилатора:

@SuppressWarnings (2/3)

41

- Всяко съобщение на компилатора принадлежи на дадена категория;
- Java Language Specification изброява два типа категории: "deprecation" и "unchecked."
 - ▣ "unchecked" предупреждение се извежда когато се използва с остарял код, написан преди въвеждането на ползата на генеричните типове (generics).
- За подтискане на повече от една категория предупреждения (warnings) се използва следният синтаксис:

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

Вътрешни анотации за компилатора - пример: `@SuppressWarnings` (3/3)

42

- От JDK 1.5 – всяка колекция трябва да бъде типизирана, противен случай излиза предупредително съобщение, като в примера – се ползва нормален списък (вместо `List<Object>`)

```
public class LegacyCode {  
    public static List array2list(Object[] array) {  
        return java.util.Arrays.asList(array);  
    }  
}
```

@SuppressWarnings - дефиниция/предварителен преглед

44

```
@Target({TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR,  
LOCAL_VARIABLE})
```

```
@Retention(RetentionPolicy.SOURCE)
```

```
public @interface SuppressWarnings {  
    String[] value();  
}
```

@SuppressWarnings – пример за употреба в метод

45

```
@SuppressWarnings("unchecked")  
public static List array2list(Object[] array) {  
    return java.util.Arrays.asList(array);  
}
```

- Компилятора **не извежда** предупреждение

@SuppressWarnings: пример за употреба в клас

46

- При имплементиране на интерфейса **Serializable** добра практика е да имаме: ***Serial Version Identifier***

```
public class WarningClass implements Serializable {  
    @SuppressWarnings("unchecked")  
    public static List toList(Object[] array) {  
        return Arrays.asList(array);  
    }  
}
```

Предупреждение : “The serializable class **WarningClass** does not declare a static final serialVersionUID field of type long”.

SuppressWarnings : serial

47

- Подтискане на предупреждението „serial”

```
@SuppressWarnings("serial")
```

```
public class WarningClass implements Serializable {  
}
```

Начин на обработка на анотации

Обработка на анотации (1 / 3)

50

- За напредналите потребители писането на анотации включва и използването на инструменти от определен тип, наречен *annotation processor*. При този тип инструмент се *чете* Java програмата и се предприемат определени действия, в зависимост от аотираните елементи и стойностите в аотациите.
- **Добра практика (пример за употреба):** посредством обработката на анотации може да се генерира се стандартен помощен код, който свързва/конфигурира текущата програма с други компоненти, следващ определени правила. По този начин се избягват множество грешки, ако този код се пише ръчно.

Обработка на анотации (2/3)

51

- **Annotation processing tool (APT)** – наличен от JDK 5.0 (с име apt)
- В JDK 6 е част от стандартният Java компилатор.

Обработка на анотации (3/3) – по време на изпълнение (runtime)

52

- За да е налична анотацията по време на изпълнение тя трябва да се анотира с анотацията `@Retention(RetentionPolicy.RUNTIME)` :

```
import java.lang.annotation.*;  
@Retention(RetentionPolicy.RUNTIME)  
@interface RuntimeAnnotation {  
}
```

Задачи за изпълнение/демо

53

Задача 1.2: Дефинирайте анотация **request-for-enhancement** (заявката за подобрение) във файл **RequestForEnhancement.java** с информация за:

- **id** (id от тип int), идентификационен номер
- **synopsis** (от тип String): причина и кратко описание на заявката за подобрение, със стойност по подразбиране "[unassigned]"
- **engineer** (от тип String): инициали на инженер
- **date** (от тип String): дата на заявката

Задачи за изпълнение/демо

54

Задача 1.3: Използвайте анотацията (**RequestForEnhancement**) от задача 1.2 в клас **TimeTravel.java** – и я приложете върху метод:

```
public static void supperFeature(Date destination) {    ...  
}
```


ANNOTATIONS (ЧАСТ 3)
ПОТРЕБИТЕЛСКИ АНОТАЦИИ И
ОБРАБОТКА

Съдържание

56

- Базов пример за дефиниране на потребителски анотации (примерно решение на задачи 1.2 и 1.3), приложение и обработка
- Стойности по подразбиране и анотация за маркиране (marker annotation)
- Елемент Value (**value ();**)
- Елементи със стойности по подразбиране
- Анотиране на анотации

Примери за потребителски анотации

Правила за методите в анотациите

58

Рядко се налага да се дефинират собствени анотации; правилата са следните:

- Подобни на интерфейсите; с водещ (@) знак преди ключовата дума за интерфейс (interface).
- Всеки от **методите** на анотациите декларира **елемент от тип за анотация**;
- Декларациите на методите не могат да имат параметри;
- Декларациите на методите не могат да хвърлят изключения;
- Типът на връщаната стойност **Е ОГРАНИЧЕН** до:
 - ▣ примитивните типове данни,
 - ▣ String,
 - ▣ Class,
 - ▣ enums,
 - ▣ annotations, и
 - ▣ Масиви от гореизброените типове;
- Методите могат да имат стойности по подразбиране (*default values*).

Примерен код към задача 1.2

59

```
/**  
 * Describes the Request-For-Enhancement(RFE) that led  
 * to the presence of the annotated API element.  
 */  
public @interface RequestForEnhancement {  
    int id();  
    String synopsis();  
    String engineer() default "[unassigned]";  
    String date() default "[unimplemented]";  
}
```

Примерен код към задача 1.3

60

- Когато анотацията е декларирана (в отделен .java файл подобно на класовете и интерфейсите) може да бъде използвана;
- Тя е специален тип модификатор и може да бъде използвана навсякъде, където останалите модификатори на езика (като например `public`, `static`, или `final`) могат да бъдат използвани.
- По конвенция анотациите предхождат останалите модификатори;
- Анотацията започва със (@) следвана от нейният тип и параметризиран списък от елементи име-стойност. Стойностите трябва да се константи по време на компилация.
- Пример за извикване на по-горната декларация:

```
@RequestForEnhancement( id = 9999, synopsis = "I need those amazing super feature",  
                        engineer = "MP", date = "26. 10. 2013")  
public static void supperFeature(Date now) {  
    // method code  
}
```

Приложение на обработката на анотации (1/2)

61

- Много приложни интерфейси (API) искат голямо количество „свързващ“ (boilerplate) код.
- Например: JAX-RPC веб услуга, ни трябва двойка интерфейс и клас имплементация;
- Те са стандартни и могат да бъдат генерирани автоматично от автоматизиран инструмент, ако програмата е анотирана с анотации, указваща кои методи ще са достъпни отдалечено;

Приложение на обработката на анотации (2/2)

62

- Други програмни интерфейси (API) искат допълнителни файлове за управление от други програми;
- Например JavaBeans иска BeanInfo class да се поддържа, докато Enterprise JavaBeans (EJB) спецификацията изисква файл т.нар. дескриптор за внедряване (*deployment descriptor*);
- По-удобно и с възможност за по-малко грешки е, ако информацията за тези допълнителни файлове се поддържат от анотации в самата програма.

Механизми за аотиране (1/2)

63

- В Java платформата винаги е имало различни механизми за аотиране;
- Например `transient` модификатора е `ad hoc` аотация, указваща при сериализация съответното поле да се пропусне;
- `@deprecated` таг от `javadoc` е `ad hoc` аотиращ елемент, че метода повече да не се използва;
- Във версия на Java 5.0, създава механизма за аотиране (познат още като метаданни - *metadata*) – възможност за обявяне на собствени типове аотации, както и синтаксис за аотиране на декларации, програмен интерфейс (API) за четене на аотации и инструмент, наречен annotation processing tool.

Механизми за аотиране (2/2)

64

- Не въздейства директно смисъла на програмата, но променят начина по който програмата се третира от инструменти и библиотеки, които могат да въздействат върху семантиката на стартираната програма.
- Анотациите могат да са налични в кода, клас файловете или по време на изпълнение.
- Анотациите допълват таговете от `javadoc`.
- В общият случай, ако аотирането е предназначено за генериране на документация, то трябва да се ползва `javadoc` таговете; в противен случай – да се ползват анотациите.

Анотации - маркери

Анотации от тип маркери

66

Определение: *Анотация, без елементи се нарича маркер.*

□ Дефиниране на маркер

*/***

** Indicates that the specification of the annotated API element*

** is preliminary and subject to change.*

**/*

public @interface Preliminary { }

□ Употреба:

@Preliminary public class TimeTravel { ... }

Елемент по подразбиране: `value()`;

67

Анотации с един елемент – името на този елемент трябва да е наречен ***value***, например:

```
/**  
 * Associates a copyright notice with the annotated API element.  
 */  
public @interface Copyright {  
    String value();  
}
```

Тогава името на елемента при извикване може да се изпусне, както и операцията присвояване (=) за такъв тип анотация:

```
@Copyright("2013 Cloudy with a Chance of Meatballs 2")  
public class ChildMovie { ... }
```


Анотиране на анотации

69

- Ключовата дума `interface` не се ползва за анотации директно.
- '@' заедно с **`interface`** на анотацията `TestAnnotation` е името на анотацията;
- Дали анотацията се прилага към клас, метод или поле (field-level annotation) е специфицирано в самата анотация. Процеса се нарича ***анотиране на анотации***.

Анотиране на анотации / Мета-анотации

70

- За област на приложение @Target
- За време на живот @Retention
- За документация @Documented

Мета-анотации : @Target

71

- В примера @TestAnnotation – се прилага само върху методи – указва се с мета анотация (метаданни за метаданните) указваща върху какви типове може да се прилага, например:@TestAnnotation

@Target(ElementType.METHOD)

```
public @interface TestAnnotation {  
    // Property Definitions here.  
}
```

Мета-анотации : @Target

72

- В примера `@TestAnnotation` е аотирана с анотацията `@Target`. Такова навързване на анотации е винаги възможно.
- В примера – новата анотация може да се използва само и единствену върху методи;
- `@Target` анотацията може да се приложи върху всеки елемент от Java, дефиниран в изброимият тип (enum) **ElementType**.

Мета-анотации: @Target / ElementType

73

- ❑ TYPE – приложимо само върху Java тип. Тип може да е Java клас, интерфейс, Enum или анотация.
- ❑ FIELD – приложими върху полета в Java (Objects, Instance or Static, декларирани на ниво клас).
- ❑ METHOD – приложимо върху метод;
- ❑ PARAMETER – приложимо само върху параметри на метод в дефиницията на метод.
- ❑ CONSTRUCTOR – върху конструктор на клас.
- ❑ LOCAL_VARIABLE – приложимо върху локални променливи (обявени в блок от код).
- ❑ ANNOTATION_TYPE – приложимо върху тип анотации.
- ❑ PACKAGE – приложимо върху пакети.

Мета-анотации: време на живот

74

- Политиката за определянето на времето на живот се задава с анотацията **@Retention**;

Мета-анотации: @Retention (1 / 2)

75

Анотация за време на живот (***Retention Policy***).

1. Само в програмният код (Source Code)
2. Също и в Class файла (освен в програмния код)
3. Run-time – налична по време на изпълнение и JVM има достъп до стойностите и

Мета-анотации: @Retention (2/2)

76

- Например: @Retention възможни стойности са съответно: SOURCE, CLASS и RUNTIME дефинирани в изброимият тип **RetentionPolicy** (Enumeration).
- Например: @TestAnnotation анотацията остава и във клас файлът
`@Target(ElementType.METHOD)`
`@Retention(RetentionPolicy.CLASS)`
`public @interface TestAnnotation {`
`// дефиниция на характеристики`
`}`

Други типове анотации: пример (1/2)

77

```
/* * @author Milen Petrov
 * @since 2011
 */
public @interface Paper {
    String leadAuthor();
    String[] authors () default {};
    String publishDate();
    String submittedDate();
    String lastAccessedDate();
    int currentRevision() default 1;
    Class<?>[] formats() default {};
    Class<? extends research.Paper>[] paperValidators() default {};
}

//Usage:
```

9.3.2017 г.

Пример за употреба (2/2)

78

```
@conference(Author = "Milen Petrov",
    value="ttt",
    Date = "17/04/2011",
    currentRevision = 2,
    LastModified = "14/02/2011",
    LastModifiedBy = "Milen Petrov",
    formats = {ConferencePaper.class, research.Citate.class},
    Reviewers = {"A.Aleksieva", "Peter", "Joan"},
    paperValidators = {
        Paper.class,
        JournalPaper.class, //extends Paper class
        ConferenceProceedingPaper.class //extends Paper class
    }
)
public class ConferencePaper {
}
```


ANNOTATIONS (ЧАСТ 4)

АНОТАЦИИ И РЕФЛЕКЦИЯ

Съдържание

80

- Механизъм на рефлексия – дефиниция, случаи на употреба
- Мета-анотации
- Пример

Потребителски случай – тестване на методи

81

- Дефиниция на анотация, чрез която ще тествахме;
- Анотиране на методи

Рефлексия: стъпка 1 (1/2)

82

Задача: *да се изгради инструмент за тестване на методи (подобно на юнит тестове). Нека декларираме анотация.*

```
import java.lang.annotation.*;  
/**  
 * върху статични методи без параметри  
 */  
@Retention(RetentionPolicy.RUNTIME)  
@Target(ElementType.METHOD)  
public @interface Test { }
```

Рефлексия: аотиране (2/2)

83

Определение: *анотация, аотирана с други анотации, наричани мета-анотации.*

- **@Retention(RetentionPolicy.RUNTIME)** индикира, че анотацията е налична за виртуалната машина по време на изпълнение;
- **@Target(ElementType.METHOD)** указва, че анотацията се използва за аотиране само на методи.

Пример : аотиране

84

```
public class HackersMovie {
```

```
    @Test
```

```
    public static void method1() { }
```

```
    public static void method2() { }
```

```
    @Test
```

```
    public static void method3() { throw new RuntimeException("Crash"); }
```

```
    public static void method4() { }
```

```
    @Test
```

```
    public static void method5() { }
```

```
    public static void method6() { }
```

```
    @Test
```

```
    public static void method7() { throw new RuntimeException("Burn"); }
```

```
    public static void method8() { }
```

```
}
```

Примерно използване на аотации

Пример : стъпка 3: Изграждане на инструмент за тестване

85

```
import java.lang.reflect.*;
public class CrashAndBurnClass {
    public static void main(String[] args) throws Exception {
        int passed = 0, failed = 0;
        for (Method method : Class.forName("HackersMovie.class").getMethods()) {
            if (method.isAnnotationPresent(Test.class)) {
                try {
                    method.invoke(null);
                    passedCounters++;
                } catch (Throwable ex) {
                    System.out.printf("Test %s failed: %s %n", method, ex.getCause());
                    failedCounters++;
                }
            }
        }
        System.out.printf("Done: %d, unfinished %d%n (crash & burn)", passedCounter, failedCounter);
    }
}
```

(чрез рефлексия)

8.3.2017 г.

Пример: стъпка 4 - резултат от стартирането (test-annotation-tool)

86

```
c:\> java CrashAndBurnClass
```

```
Test public static void CrashAndBurnClass.method3() failed:  
java.lang.RuntimeException: Crash
```

```
Test public static void CrashAndBurnClass.method7() failed:  
java.lang.RuntimeException: Burn
```

```
Done: 2, Unfinished 2 (crash & burn)
```


Примери

87

```
final class Employee {  
    private String name;  
    private String id;  
    // set/get методи.  
}
```

Анотация за тестване

TestAnnotation.java

```
public @interface TestAnnotation {  
    //дефиниция на свойства  
}
```

ANNOTATIONS – НОВОСТИ В JAVA 8

Milen Petrov
2014

Съдържание

89

- Предпоставки за развитие
- Повтарящи се анотации

Предпоставки за развитие на анотациите

90

- Развитие на инструментите
- Развитие на езика и Java екосистемата
- Развитие на свързаните технологии

Type annotations(**NEW! Java SE 8**) (1/2)

- **Анотациите** могат да се прилагат и при използване на типовете (***use of types***)

Използване на анотации (Java SE 8) (2/2)

1. Създаване на обект `new @Interned MyObject();`
2. Преобразуване на типове (type cast): `myString = (@NonNull String) str;`
3. При клаузата `implements`: `class`
`UnmodifiableList<T> implements @ReadOnly List<@ReadOnly T> {`
`... }`
4. При хвърляне на изключения: `void`
`folderMonitor () throws @Critical InvalidFolderException { ... }`

Type Annotations and Pluggable Type Systems

- От Java SE 8+ анотации, освен в декларациите могат да се използват и на всякъде, където могат да се използват и типове (например при new);
- Целта е по-силна типизация и проверка на типовете;
- Може да създадете собствена система за типизация (или да ползвате готова) – посредством приставки (pluggable)
- **@NonNull** String str; (например може да се ползва на готово система от тук: <http://types.cs.washington.edu/checker-framework/>)

References

- Type Annotations and Pluggable Type Systems.

Повтарящи се анотации (Repeating annotations (**NEW! Java SE 8**): Използване

Пример:

```
@Lector(name = "Milen Petrov")
```

```
@Lector(name = "Adelina Aleksieva")
```

```
class ClassJavaEE { ... }
```

- Върху класове
- Върху методи
- За обратна съвместимост повтарящите се анотации се съхраняват в автоматично генериран от компилатора контейнер за анотации (*container annotation*)

@Repeatable meta-annotation

деклариране (1/2)

□ `import java.lang.annotation.Repeatable;`

Стъпка 1:

□ `@Repeatable(Lectures.class)`

```
public @interface Lecture {  
    String name() default "NA";  
    int mainLecture() default 0;  
}
```

Заб. 1) В `Lectures.class` е контейнера за повтарящите се анотации

2) При повторение на анотации, недекларирани като `@Repeatable` – ще се генерира грешка по време на компилация

Анотация-контейнер (2/2)

Стъпка 2: Масив от повтарящата се анотацията

```
public @interface Lectures {  
    Lecture[] value();  
}
```

Рефлексия за повтарящи се анотации

- Методи за единични анотации при рефлексията са непроменени (като например [AnnotatedElement.getAnnotationByType\(Class<T>\)](#)), ако се изисква само един тип.
- Ако се искат повече анотации – трябва да се извлече първо анотацията контейнер и после да се извечат повтарящите се анотации (за обратна съвместимост със стар код)
- Новите методи в Java SE 8 имат функционалност (като например [AnnotatedElement.getAnnotations\(Class<T>\)](#)) – да връщат множество анотации
- За допълнителна информация:
<http://docs.oracle.com/javase/8/docs/api/java/lang/reflect/AnnotatedElement.html>

Решения при проектиране на използването на анотации

- Трябва да се проектира кардиналността (cardinality) на анотациите на зададения тип;
- Дадена анотация може да се прилага
 - ▣ 0
 - ▣ 1
 - ▣ Повече пъти (@Repeatable)
- Може да се ограничи прилагането посредством анотацията @Target; (например да се ползва само за полета и методи);
- Трябва да се проектират, така че за програмистът, ползващ анотацията да е достатъчно гъвкава и мощна за прилагане;

Други анотации

- **@SafeVarargs** анотация когато се приложи до метод или конструктор проверява, че кода не извършва потенциално ,несигурни‘ операции върху своите променлив брой аргументи (varargs); Когато е използван тип анотация - за непроверени предупреждения (unchecked warnings) – то те се подтискат (suppress) – от JDK 7
- **@FunctionalInterface** (Java SE 8), указва, че дадената анотация е за функционален интерфейс.

Методи с дефиниция по подразбиране (Java 8)

```
public interface iEosStudent extends iStudent {  
    // void study(); // from iStudent  
    // void studyNow(); // from iStudent  
    void mmStudy(); //old method  
    default boolean studyNow() {  
        // System.out.println("study mumbo-jumbo");  
    }  
}
```

Правило: класове, използващи интерфейси - с добавени методи по подразбиране – не е необходимо да бъдат прекомпилирани

Lambdas (Java 8)

- <http://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>
- (TODO: move to netJava)

Благодаря за вниманието

105

9.3.2017 г.