

Annotations en Java

Université de Montpellier
Java avancé (IG4)

2019

Annotations

- Méta-données ou Informations ajoutées aux programmes pour leur traitement par des outils
 - éditeurs
 - débogueurs
 - générateurs de documentation (tags Javadoc)
 - outils de vérification
 - outils de test, statistiques, refactoring, etc.

Utilisation dans javadoc

```
/**  
 * @deprecated As of JDK version 1.1,  
 * replaced by Calendar.get(Calendar.MONTH)  
 * as shown in {@link java.util.Calendar#get(int) get}  
 */
```

Produit dans la documentation :

getMonth

public int getMonth()

Deprecated.

***As of JDK version 1.1, replaced by
Calendar.get(Calendar.MONTH) as shown in [get](#)***

Utilisation dans javadoc

```
/**  
 * @deprecated As of JDK version 1.1,  
 * replaced by Calendar.get(Calendar.MONTH)  
 * as shown in {@link java.util.Calendar#get(int) get}  
 */
```

- block tag `@deprecated` en début de ligne
- inline tag `{@link }` en milieu de ligne
- utilisé par le programme javadoc pour créer les pages html de la documentation

Utilisation dans Eclipse

Une méthode privée inutilisée génère un warning

```
private void crypter()  
{/* à écrire plus tard*/ System.out.println("cryptage");}
```

Solution proposée par Eclipse

ajouter un tag pour faire disparaître ce warning

```
@SuppressWarnings("unused")  
private void crypter()  
{/* à écrire plus tard*/ System.out.println("cryptage");}
```

l'IDE n'affichera plus de warning !

Utilisation dans Eclipse

Pour indiquer au compilateur de vérifier la signature d'une opération lors de sa redéfinition dans une sous-classe

```
public class Personnage {  
    .....  
    @Override  
    public boolean equals(Personnage p) {  
        return ....;  
    }  
}
```

l'IDE signalera l'erreur "must override or implement a supertype method"

Correction : `public boolean equals(Object p)`

Déclaration d'un type d'annotation

- type d'annotation = interface
- mot-clef *interface* précédé par @
- les méthodes
 - définissent des éléments
 - quand il est unique, l'élément s'appelle *value* (et son nom peut être omis)
 - pas de paramètre
 - pas de clause throws
 - type de retour possible (TRP)
 - TRP = types primitifs, String, Class, enum,
 - arrays de TRP
 - valeurs par défaut

Définition d'un type d'annotation

```
/**  
 * Request-For-Enhancement(RFE)  
 * pour annoter un élément à améliorer  
 * dans la version suivante  
 */  
public @interface RequestForEnhancement  
{  
    int id();  
    String synopsis();  
    String engineer() default "[unassigned]";  
    String date() default "[unimplemented]";  
}
```


Utilisation de l'annotation

~ se place comme un *modifier*

```
@RequestForEnhancement(  
    id = 23777,  
    synopsis = "Improve time complexity",  
    engineer = "Jack",  
    date = "31 oct 2009")
```

```
public static
```

```
<T extends Comparable<? super T>>
```

```
void sort(List<T> list)
```

```
{ ... }
```

interface **Annotation**

- C'est l'interface spécialisée par les annotations
- Ne pas l'étendre manuellement

- Méthodes liées à l'introspection

Class<? extends **Annotation**> **annotationType**()
retourne le type d'annotation de cette annotation

boolean **equals**(**Object** obj)

int **hashCode**()

String toString()

Types d'annotations de l'API

- Annotations
 - Deprecated
 - Override
 - SuppressWarnings
- Certaines portent sur d'autres annotations
 - Inherited
 - Documented
 - Repeatable
 - Retention : décrit la portée
 - SOURCE, CLASS, RUNTIME
 - Target : décrit la cible
 - TYPE, FIELD, METHOD, ANNOTATION_TYPE etc.

Exemple d'annotation de l'API elle-même annotée

@Documented

@Retention(value=RUNTIME)

@Target(value=ANNOTATION_TYPE)

public @interface **Retention**

{ **RetentionPolicy** value(); }

Interface **AnnotatedElement**

- Pour observer les éléments annotés
- Implémentée par `AccessibleObject`, `Class`, `Constructor`, `Field`, `Method`, `Package`

Méthodes

`<T extends Annotation> getAnnotation`
`(Class<T> annotationType)`

retourne l'annotation attachée dont le type est passé en paramètre (ou null)

`Annotation[] getAnnotations` ()

retourne les annotations attachées à l'élément (incluant héritées)

`Annotation[] getDeclaredAnnotations` ()

retourne toutes les annotations attachées à l'élément (propres)

`boolean isAnnotationPresent`

`(Class<? extends Annotation>annotationType)`

retourne vrai ssi une annotation du type passé en paramètre est attachée à l'élément

Exemple d'utilisation

Éléments pour un outil de test

- Objectif :
 - embarquer dans les classes des méthodes de test unitaire
 - annotation par les programmeurs de ces méthodes de test (pour les distinguer des autres)
 - l'outil de test utilise les annotations pour tester la classe

Type d'annotation pour les méthodes de test

```
import java.lang.annotation.*;
enum NiveauRisque {faible, moyen, eleve;}

/**
 * indique qu'une méthode est une méthode de test
 * à utiliser sur des méthodes sans paramètre
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test
    {NiveauRisque risque();}
```

Une classe en développement

```
class Foo {  
  
    @Test(risque=NiveauRisque.faible)  
        public static void m1() {System.out.println("m1");}  
  
    public static void m2() {System.out.println("m2");}  
  
    @Test(risque=NiveauRisque.moyen)  
        public static void m3() {throw new RuntimeException("Boom");}  
  
    public static void m4() {System.out.println("m4");}  
  
    @Test(risque=NiveauRisque.moyen)  
        public static void m5() {System.out.println("m5");}  
  
    public static void m6() {System.out.println("m6");}  
  
    @Test(risque=NiveauRisque.eleve)  
        public static void m7() {throw new RuntimeException("Crash");}  
  
    public static void m8() {System.out.println("m7");}  
  
}
```


Une classe de l'outil de test

```
import java.lang.annotation.*;
import java.lang.reflect.*;
public class TestAnnotations
{
    public static void main(String[] className) throws Exception
    {
        int passed = 0, failed = 0;
        for (Method m : Class.forName(args[0]).getMethods()) {
            if (m.isAnnotationPresent(Test.class) &&
                (m.getAnnotation(Test.class).risque() != NiveauRisque.faible))
            {
                try {m.invoke(null); passed++; }
                catch (Throwable ex)
                {
                    System.out.println("Test "+m+" failed:"+ex.getCause());
                    failed++;
                }
            }
        }
        System.out.println("Passed: "+passed+" Failed "+failed);
    }
}
```

Exécution

Pour tester la classe Foo

Prompt> java TestAnnotations Foo

```
>> Test public static void Foo.m3() failed: java.lang.RuntimeException: Boom  
>> m5  
>> Test public static void Foo.m7() failed: java.lang.RuntimeException: Crash  
>> Passed: 1 Failed 2
```

Nota: pour simplifier, les méthodes de Foo sont statiques et sans paramètre mais elles pourraient être appliquées à des instances et avec des paramètres

Composer des annotations

- une annotation ne peut en spécialiser une autre

```
public @interface ClassInfo{  
    String createur();  
    String testeur()  
}
```

```
public @interface ClassInfoVersion extends ClassInfo  
{int version();}
```

Composer des annotations

```
public @interface ClassInfo{  
    String createur();  
    String testeur();  
}
```

```
@interface Version  
{int version();}
```

```
@interface ClassInfoVersion{  
    ClassInfo classinfo();  
    Version version();  
}
```

Synthèse

- **Introspection Java :**
 - un mécanisme pour interroger le programme pendant l'exécution (runtime)
 - Un aspect de la méta-programmation
- **Annotations :**
 - Méta-données placées dans le code source
 - Destinées
 - au compilateur,
 - aux outils de documentation ou de vérification
 - à la machine virtuelle