Annotations en Java

Université de Montpellier

HAI401I – MPO2

2022

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
- beaucoup d'autres tags pour générer des documentations complètes

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

- Interface de l'API Java
- C'est l'interface spécialisée par les annotations
- Ne pas l'étendre manuellement

Types d'annotations de l'API

- Annotations
  - Deprecated
  - Override
- 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.

Types d’annotations de l’API

Retention : décrit la portée

- SOURCE
  - écartées par le compilateur
  - Utilisées par les outils manipulant le code source (.java)
- CLASS
  - Utilisées par le compilateur (.class)
  - peuvent être écartées pour la machine virtuelle
- RUNTIME
  - Utilisées par le compilateur (.class)
  - Utilisées par la machine virtuelle

Exemple d'utilisation
Eléments 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”);}
}
```

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

- Annotations :
  - Méta-données placées dans le code source
  - Destinées
    - au compilateur,
    - aux outils de documentation ou de vérification
    - à la machine virtuelle

- On en manipulera certaines par introspection !