Les annotations

contenu de la section

Quelques généralités

les meta-données

- données sur les données ou sur les programmes qui n'affectent pas directement la sémantique du programme
- documentation, test à la compilation (@override), directives de configuration, analyse de code (dépendances), directive d'optimisation à l'exécution
- les méta-données peuvent concerner des outils externes (javadoc) ou la JVM

directive pour autoriser la sérialisation, le clonage, ...

l'objectif des annotations

- standardiser la définition et l'usage des méta-données associées aux programmes java
- rapprocher les méta-données des données (présence dans le même fichier)
- permettre l'accès aux méta-donnés au niveau des fichiers sources, des fichiers .class ou à l'exécution
- les annotations complémentent les tags javadoc mais ne les remplacent pas !!!

les annotations et les environnements java

- Les annotations sont très utilisées aujourd'hui :
  - pour la configuration des serveurs d'applications (JBoss, Jonas, ….)
  - dans les frameworks : Spring, Hibernate, EJB (@Entity, @Stateless, ...), junit (@test, ...) ….

L'utilisation des annotations

la syntaxe d'usage

type d'annotation vs annotation (équivalent à class vs instance)

- une annotation consiste en @ suivi du type de l'annotation et d'une liste parenthésée d'éléments : nom= valeur
- un type d'annotation sans élément est appelé type marqueur
- le nom de l'élément dans un type d'annotation n'ayant qu'un élément unique est nécessairement value

- une annotation est considérée comme un modificateur
- par convention une annotation précède les autres modificateurs dans le code

forme abrégée courante pour la forme complète :**@Version(value="3.0.2.alpha")**

```
.....
@Version("3.0.2.alpha")
public class AnnotExample1 {
        .....
        @TODO(importance=5,developper="fho",date="01/04/2005")
        public void m1(int x, boolean y) { /*A FAIRE */ }

        @TODO(importance=3,description="utiliser l'algo de xyz")
        public void m2(int x) { /*A FAIRE */ }

        @Deprecated
        public void m3(…….) { …… }
        …….
```

une annotation du type Version

2 annotations de type TODO

annotation du type marqueur Deprecated

La définition d'un nouveau type d'annotation (1)

la syntaxe de définition

- semblable à une définition d'interface mais …
  - @ précède le mot clé interface
  - chaque élément du type d'annotation est défini par une méthode sans argument, ni clause throws
  - les types de retour valides : les types primitifs, String, les énumérations, les annotations et les tableaux des types précédents
  - possibilité de définir une valeur par défaut (via le mot clé **default**)

un exemple simple

définition d'un type d'annotation

```
package tiger.annotations;

public @interface TODO {
        int importance();
        String description() default "";
        String developper() default "unassigned";
        String date() default "not done";
}
```

création d'une annotation

```
package tiger.examples;

public class AnnotExample1 {

   @TODO(importance=5,developper="fho",date="01/04/2005")
   public void m1(int x, boolean y) { /*A FAIRE */}

   @TODO(importance=3,description="utiliser l'algo de xyz")
   public void m2(int x) { /*A FAIRE */}

   ……
```

Compléments sur la définition d'un nouveau type d'annotation (2)

les éléments composites

- les éléments de type tableau
- les références à d'autres annotations

```
package tiger.examples;

public interface AnnotExample2 {

    @Annotation2(infos= { "A", "B","C","D"} ,anno=@Annotation1("E"))
    public void m1(int x, boolean y);

    @Annotation2(infos="A",anno=@Annotation1("E"))
    public void m2(int x, boolean y);

}
```

```
public @interface Annotation1 {
        String value() default "" ;
}
```

```
public @interface Annotation2 {
        String[ ] infos() default { "NONE" };
        Annotation1 anno();
}
```

les annotations prédéfinies

- **@Override**
- **@Deprecated**
- **@SuppressWarnings("designation du warning")**

uniquement visible dans les sources (cf. suite)

La définition d'un nouveau type d'annotation (3)

les méta-annotations

- @Retention a un élément de type enum RetentionPolicy

RetentionPolicy.SOURCE	l'annotation n'est pas stockée dans le fichier .class
RetentionPolicy.CLASS	l'annotation est stockée dans le fichier .class mais est ignorée par la JVM
RetentionPolicy.RUNTIME	l'annotation est stockée dans le fichier .class et est lue par la JVM

- @Target a un élément de type tableau de enum ElemType

ElementType.TYPE	les cibles de l'annotation sont les classes, les interfaces, les enum
ElementType.FIELD	les cibles de l'annotation sont les champs, les valeurs des enum
ElementType.METHOD	les cibles de l'annotation sont les méthodes (sans les constructeurs)
ElementType.PARAMETER	les cibles de l'annotation sont les paramètres de methodes
ElementType.CONSTRUCTOR	les cibles de l'annotation sont les constructeurs
ElementType.LOCAL_VARIABLE	les cibles de l'annotation sont les variables locales
ElementType.ANNOTATION_TYPE	les cibles de l'annotation sont les annotations (méta-annotations)
ElementType.PACKAGE	les cibles de l'annotation sont les packages

- @Documented marqueur indiquant que l'annotation doit apparaître dans la javadoc
- @Inherited marqueur indiquant que l'annotation est héritée par les descendants de la classe annotée

exigent @Retention(RetentionPolicy.RUNTIME)

L'utilisation d'un nouveau type d'annotation (1)

l'accès réflexif aux annotations

- nouvelles méthodes offertes par **Class**

```
Annotation[] getAnnotations()
Annotation getAnnotation(Class annotation_type)
Annotation[] getDeclaredAnnotations()

boolean isAnnotation()
boolean isAnnotationPresent(Class annotation_type)
```

- nouvelles méthodes offertes par les classes du package java.lang.reflect

```
Annotation[] getAnnotations()
Annotation getAnnotation(Class annotation_type)
Annotation[] getDeclaredAnnotations()

boolean isAnnotationPresent(Class annotation_type)
```

L'utilisation d'un nouveau type d'annotation (2)

un exemple simple ….

```
public class TODOTest {
    public static void main(String[] args) {
        if (args == null || args[0] == null) return;
        try {
            Method[] meths = Class.forName(args[0]).getMethods();
            for (int i = 0; i< meths.length; i++) {
                if (meths[i].isAnnotationPresent(tiger.annotations.TODO.class)) {
                    Annotation annot = meths[i].getAnnotation(tiger.annotations.TODO.class);
                    System.out.print("la methode " + meths[i].toString() + " est annotee par " + annot.toString());

                }
            } catch (SecurityException e) {
                e.printStackTrace();
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
    }
}
```

indispensable pour pouvoir récupérer l'annotation à l'exécution

```
package tiger.annotations;
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)

public @interface TODO {
    int importance();
    String description() default "";
    String developper() default "unassigned";
    String date() default "not done";
}
```