

# 11. Les annotations

Annotation = meta-tag


*Programmation déclarative:*

le programmeur dit ce qu'il faut faire et des outils génèrent le code pour le faire

Le compilateur ou la machine virtuelle extraient, à partir des annotations, des informations sur le comportement du programme

## Déclaration d'un type d'annotation:

```
public @interface AnnotationName  
    { ... method1() ... ;  
      ... method2() ... ;  
      ... method3() ... ;  
      ...  
    }
```



Déclarations de méthodes

## Utilisation: annoter un code

```
@AnnotationName ( method1 = value1,  
                   method2 = value2,  
                   method3 = value3, ... )
```

Code que l'on peut annoter:

- **Package**
- **Classe, interface, annotation, ...**
- **Variable d'instance**
- **Constructeur**
- **Méthodes**
- **Variables**
- **Arguments**

## Catégories d'annotations

### 1. Marker : sans élément

Ex: public @*interface* MyMarker {  
}

## Exemples d'utilisation de cette annotation:

**@MyMarker**  *Annote une classe*

```
public class MyClass {
```

**@MyMarker**  *Annote une variable d'instance*

```
    private int var1;
```

**@MyMarker**  *Annote un constructeur*

```
    public MyClass(int i)    { var1 = i;}
```

**@MyMarker**  *Annote une méthode*

```
    public void myMethod1(int a) { var1 +=a;}
```

```
}
```

## 2. Single-element annotation

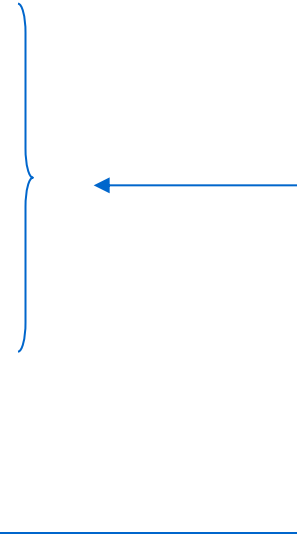
Ex: public @*interface* MySingleValueAnnotation

```
{ String value() default "bidon";  
}
```

  
facultatif

## Règle de déclaration des méthodes dans les annotations:

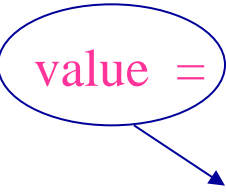
- Pas de paramètre
- Pas de clause throws
- Types de retour: uniquement
  - Types primitifs (int, float, char, ...)
  - String
  - Class
  - Énumération
- + Tableau de types ci-dessus





## Exemples d'utilisation de cette annotation:

```
@MySingleValueAnnotation ( value = "test" )  
public class MyClass {
```



*Pas obligatoire si annotation  
single-element*

```
@MySingleValueAnnotation ("test") private int var1;
```

```
@MySingleValueAnnotation ("test")
```

```
public MyClass(int i) { var1 = i;}
```

```
@MySingleValueAnnotation ("test")
```

```
public void myMethod1(int a) { var1 +=a;}
```

```
}
```

### 3. Multi-value annotation

Ex:

Soit l'énumération:

```
public enum MonthEnum
```

```
    {JANUARY, MARCH, MAY, JULY, SEPTEMBER, NOVEMBER}
```

```
public @interface MyMultiValueAnnotation {
```

```
    public int myIntValue( ) default 24;
```

```
    public String myStringValue( ) default "now";
```

```
    public String[] myTabValue( ) default {"Monday,Tuesday"};
```

```
    public MonthEnum myEnumValue( ) default MonthEnum.JULY;
```

```
}
```

Exemples d'utilisation de cette annotation:

`@MyMultiValueAnnotation`

```
public class MyClass {
```

```
@MyMultiValueAnnotation (myIntValue = 12) private int var1;
```

```
@MyMultiValueAnnotation ( myTabValue = { "friday", "saturday", "sunday" })
```

```
public MyClass(int i) { var1 = i;}
```

```
@MyMultiValueAnnotation ( myTabValue = { "friday", "saturday", "sunday" },
```

```
    myEnumValue = MonthEnum.MAY)
```

```
public void myMethod1(int a) { var1 +=a;}
```

```
}
```

## Build-in annotations

### 1. Annotations simples:

- **Override**

- **Deprecated**

- **SuppressWarnings**

## 1. Override

Une méthode annotée *Override* doit redéfinir une méthode de la super-classe

Ex:

```
public class MyClass {  
    @Override public String toString( )  
        {return "Blabla";}  
}
```

Si la méthode annotée *Override* n'existe pas dans la super-classe, il y a erreur à la compilation

## 2. Deprecated

Ex:

```
public class MyClass {  
    @Deprecated public int doSomething()  
        {return 0;}  
}
```

Si appel de la méthode doSomething( ) :

à la compilation:


avertissement signalant que la méthode est déconseillée car obsolète

### 3. SuppressWarnings

Permet de supprimer des avertissements à la compilation

Ex:

```
public class Principal {  
    @SuppressWarnings ({"deprecation"})  
    public static void main(String[] args)  
    {  
        MyClass m = new MyClass( );  
        System.out.println(m.doSomething( ) );  
    }  
}
```

  
*deprecated*

Si la méthode doSomething( ) de la classe MyClass est « *deprecated* », le compilateur n'affichera pas d'avertissement.

## **2. Meta-annotations**

**= Annotation de type d'annotation**

- Target**
- Retention**
- Documented**
- Inherited**



## 1. Target

Permet de préciser sur quel type d'élément l'annotation peut porter.

Types d'éléments:

ElementType.**TYPE**

ElementType.**FIELD**

ElementType.**METHOD**

ElementType.**PARAMETER**

ElementType.**CONSTRUCTOR**

ElementType.**LOCAL\_VARIABLE**

ElementType.**ANNOTATION\_TYPE**

Ex:

```
import java.lang.annotation.*;
```

```
@Target( { ElementType.CONSTRUCTOR, ElementType.METHOD } )
```

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

Si on tente d'appliquer cette annotation à d'autres éléments qu'un constructeur ou une méthode, il y a *erreur à la compilation*

## Exemples d'utilisation de cette annotation:

`@MySingleValueAnnotation`  *Erreur à la compilation*

```
public class MyClass {
```

`@MySingleValueAnnotation ("test")`  *Erreur à la compilation*

```
private int var1;
```

`@MySingleValueAnnotation ("test")`

```
public MyClass(int i) { var1 = i;}
```

`@MySingleValueAnnotation ("test")`

```
public void myMethod1(int a) { var1 +=a;}
```

```
}
```

OK

## 2. Retention

Permet de préciser le niveau auquel est « retenue » l'annotation.

Trois niveaux:

RetentionPolicy.**SOURCE**:

*l'annotation sera retenue uniquement au niveau code source et ignorée par le compilateur*

RetentionPolicy.**CLASS**:

*l'annotation sera retenue par le compilateur à la compilation mais ignorée par la machine virtuelle*

RetentionPolicy.**RUNTIME**:

*l'annotation sera retenue par la machine virtuelle et ne pourra être lue qu'à l'exécution*

Ex:

```
import java.lang.annotation.*;
```

**@Retention** (RetentionPolicy.*RUNTIME*)

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

### 3. Documented

Permet d'inclure l'annotation dans la javadoc.

Ex:

@Documented

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

## 4. Inherited

Seulement pour les annotations sur des **classes**

Si une annotation A annotée @Inherited porte sur une classe C, alors toute **sous-classe de la classe C en héritera**, c'est-à-dire, sera également annotée par l'annotation A.

Ex:

**@Inherited**

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

Exemple d'utilisation de cette annotation:

**@MySingleValueAnnotation**

```
public class MyClass { ... }
```

⇒ Pour toute sous-classe de MyClass:

**@MySingleValueAnnotation**

```
public class MySubClass extends MyClass { ... }
```



## JavaDoc

Les commentaires placés entre

/\*\*

\*

\*

\*/

seront repris dans la JavaDoc (générée automatiquement)

## Quelques annotations utilisables dans la JavaDoc

### Documentation d'une classe:

@author

@version

@since

@see *Suggestion de consulter d'autres sources, ...*

*ex:    @see #field        (si dans même classe)*

*@see #Constructor(type, type, ...)*

*@see #method(type, type, ...)*

*@see Class        (si dans même package)*

*@see package.Class*

*@see package.Class#field*

*@see package.Class#method(type, type, ...)*

## Documentation d'une méthode/constructeur:

@param *paramName*

*Informations sur les paramètres*

@return

*Informations sur la valeur de retour*

@throws

*Informations sur les exceptions détectées*

@see

...