Les annotations Java

Jonathan Lejeune

Sorbonne Université/LIP6-INRIA

SRCS – Master 1 SAR 2019/2020

sources :

Développons en Java, Jean-Michel Doudoux

Les annotations

Définition

Méta-données sur une entité d'un programme Java, pouvant être exploitées à la compilation et/ou à l'exécution

Utilisations

- Détection d'erreur à la compilation
- Génération de code (création de nouveaux fichiers sources)
- Génération de fichiers (ex : fichier de configuration, de documentation)

$\Rightarrow$ Simplification de développement, code allégé

Quelques API utilisant les annotations

- Javadoc et Jdk standard : `@Deprecated`, `@Override`, `@SuppressWarning`
- Junit 4 : `@Test`, `@Ignore`, `@Before`, `@After`, ...
- EJB 3 : `@Remote`, `@Stateless`, `@Entity`, ...

Mise en œuvre des annotations

Appel à une annotation

- Caractère `@` + nom de l'annotation
- Doit obligatoirement précéder l'entité qu'elle annote
- Sur une ligne dédiée par convention

Associations possibles à toute entité Java

- packages, classes, interfaces
- constructeurs, méthodes, attributs
- paramètres, variables locales
- ou des annotations (méta-annotation)

Paramétrage des annotations

Catégories d'annotation

- les **marqueurs** : annotations sans attributs

  ex : `@Override`
- les **annotations paramétrées** : un seul attribut

  ex : `@MonAnnot("test")`
- les **annotations multi-paramétrées** : plusieurs attributs

  ex : `@MonAnnot(arg1="test1", arg2="test2",arg3="test3")`

Types possibles des attributs

- Chaîne de caractère : `String`
- type primitif : `int`, `boolean`, `double`, ...
- énumération :

  `enum couleur{ ROUGE, VERT, BLEU}`
- annotation
- le type `class` (réflexivité Java)

} + tableau java de ces types

Les annotations standards : `@Deprecated`

Caractéristiques

- indique qu'une entité ne devrait plus être utilisée
  $\Rightarrow$ warning à la compilation sinon
- pas d'argument
- s'applique sur des classes, interfaces ou membre (méthode ou attribut)
- à ne pas confondre avec `@deprecated` de la javadoc

```
class Truc{

 /**
 * Fait quelque chose
 * @deprecated plus compatible
 */
 @Deprecated
 public void f(){..}
}
```

Les annotations standards : @Override

Caractéristiques

- indique qu'une méthode est une redéfinition d'une méthode héritée
  $\Rightarrow$ erreur si la méthode n'existe pas dans les types parents
- pas d'argument
- s'applique sur des méthodes

```
public class Mere{

 @Override //OK car heritage de Object
 public String toString(){..}
}

public class Fille extends Mere{
 @Override // erreur compilation car f n'existe pas dans Mere
 public void f(){..}
}
```

Les annotations standards : `@SuppressWarning`

Caractéristiques

- permet de demander au compilo d'hiniber certains avertissements qui sont pris en compte par défaut
- Argument de type `String` représentant un warning à ignorer :
  - `"deprecation"` : utilisation d'une méthode dépréciée
  - `"unchecked"` : pas d'utilisation des génériques sur une collection
  - `"fallthrough"` : pas de `break` dans chaque `case` d'un `switch`
  - `"path"` : erreur sur le chemin fournis en paramètre du compilo
  - `"serial"` : absence d'un serialVersionUID dans les bean
  - `"finally"` : `return` dans un `finally`
  - `"unused"` : une variable locale ou champs privé qui n'est jamais utilisé

Il vaut mieux essayer de résoudre l'avertissement plutôt que d'utiliser cette annotation

Annotations personnalisées

Déclaration

Dans un fichier MonAnnotation.java

```
package com.lejeune.test.annotations;

public @interface MonAnnotation{
    String arg1();
    int arg2();
    boolean arg3() default true; //valeur par defaut
    String[] arg4() default {"v1","v2"};
}
```

Utilisation

```
package un.autre.endroit;
import com.lejeune.test.annotations.*;

@MonAnnotation(arg1="valeur1", arg2=4)
```

La méta-annotation @Target

Caractéristiques

- Précise les entités sur lesquelles l'annotation peut être utilisée.
- Un seul attribut : un tableau de valeurs issues de l'enum `ElementType` :
  - `CONSTRUCTOR` : applicable aux constructeurs
  - `LOCAL_VARIABLE` : applicable aux variables locales
  - `FIELD` : applicable aux attributs
  - `METHOD` : applicable aux méthodes
  - `PACKAGE` : applicable aux packages
  - `PARAMETER` : applicable aux paramètres d'une méthode ou constructeur
  - `TYPE` : applicable aux classes, interfaces, énumérations, annotations
  - `ANNOTATION_TYPE` : applicable uniquement aux annotations

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

@MonAnnotation(arg1="valeur1", arg2=4) // ERREUR
public class MaClasse{..}
```

La méta-annotations `@Retention`

Caractéristiques

- Précise à quel niveau les infos de l'annotation seront conservées.
- Un seul attribut : une valeur de l'enum `RetentionPolicy` :
  - `SOURCE` : info conservées uniquement dans le code source
    $\Rightarrow$ Le compilateur les ignore
  - `CLASS` : info conservées dans le source et le bytecode
  - `RUNTIME` : info conservées dans le source et le byte code **et** disponibles à l'exécution par introspection

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD,ElementType.CONSTRUCTOR})
public @interface MonAnnotation{...}
```

Les méta-annotations @Inherited et @Documented

@Inherited

Permet d'appliquer l'annotation à une classe et automatiquement à l'ensemble de ses classes filles directes et indirectes

```
@Target(ElementType.TYPE)
@Inherited
public @interface MonAnnotation{...}

@MonAnnotation(arg1="valeur1", arg2=4)
public class Mere{...}

// implicitement @MonAnnotation(arg1="valeur1", arg2=4)
public class Fille extends Mere{...}
```

@Documented

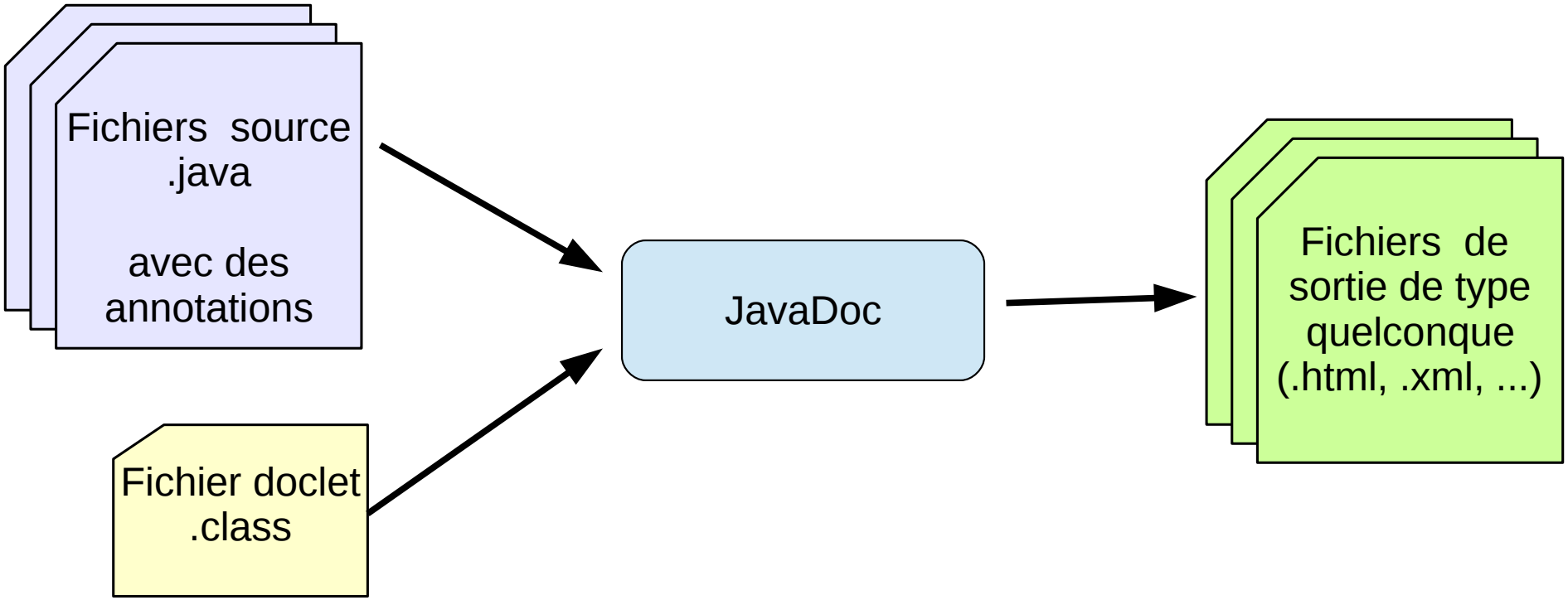
Indique qu'il faut intégrer la définition de l'annotation dans la documentation générée par JavaDoc

Exploitation des Annotations

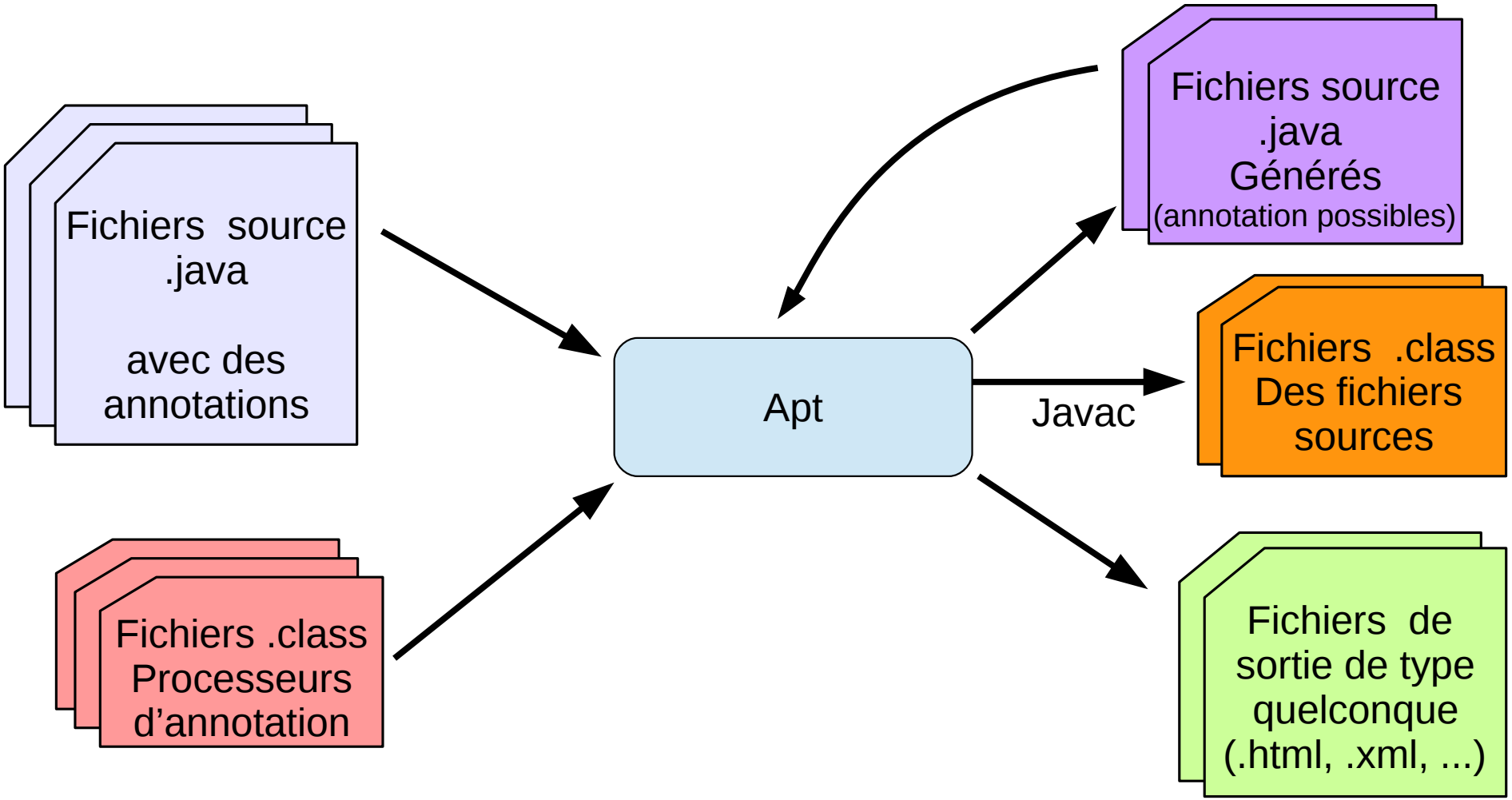
Plusieurs outils

- Analyseur du code source : **Javadoc + définition de doclet**
- Exploitation au moment de la compilation : **Apt**
- Exploitation au moment de l'exécution : **introspection Java**

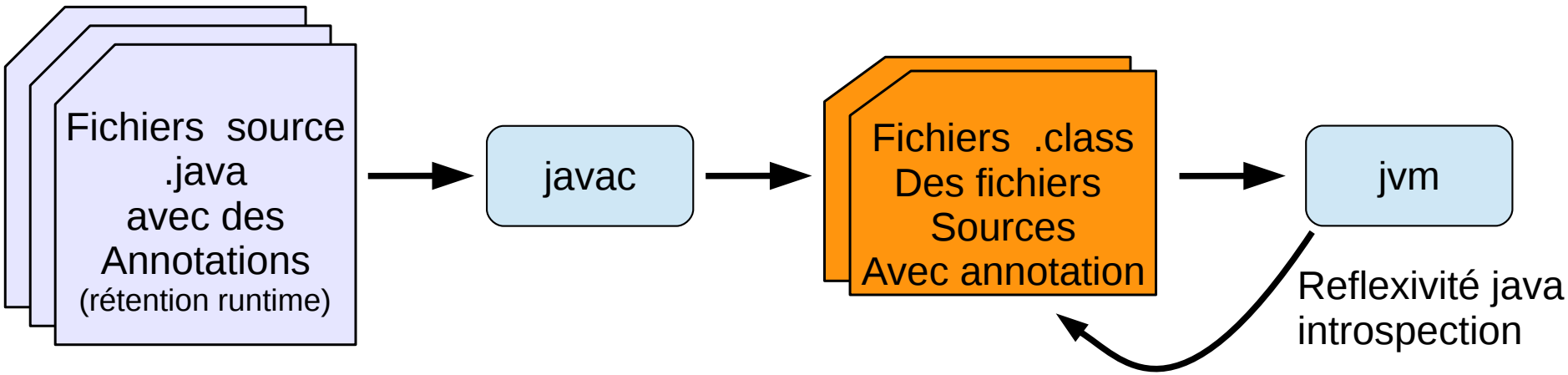
Analyseur de code source



Exploitation au moment de la compilation



Exploitation à l'exécution




```
package java.lang.reflect;

public interface AnnotatedElement{
 <T extends Annotation> getAnnotation(Class<T>);
 Annotation[] getAnnotations();
 Annotation[] getDeclaredAnnotations();
}
```

Types réflexifs implantant `AnnotedElement` :

`AccessibleObject`, `Class`, `Constructor`, `Field`, `Method` et `Package`