Exemples du cours 2 de Ingénierie du logiciel M2 ISIN

F. Gava

Pour suivre le cours

1 SimpleAnnotationProcessorFactory.java

```
public class SimpleAnnotationProcessorFactory implements AnnotationProcessorFactory {
 /** Collection contenant le nom des Annotations supportées. */
 protected Collection<String> supportedAnnotationTypes =
       Arrays.asList( TODO.class.getName(), TODOs.class.getName() );

 /** Collection des options supportées. */
 protected Collection<String> supportedOptions = Collections.emptyList();

 /** Retourne la liste des annotations supportées par cette Factory. */
 public Collection<String> supportedAnnotationTypes() {return supportedAnnotationTypes;}

 /** Retourne la liste des options supportées par cette Factory. */
  public Collection<String> supportedOptions() {return supportedOptions;}

 /** Retourne l'AnnotationProcessor associé avec cette Factory. */
  public AnnotationProcessor getProcessorFor(Set<AnnotationTypeDeclaration> atds,
              AnnotationProcessorEnvironment env) {
     // Si aucune annotation n'est présente on retourne un processeur "vide"
     if (atds.isEmpty())
      return AnnotationProcessors.NO_OP;
      return new SimpleAnnotationProcessor(env);
}}
```

Si **apt** trouve des annotations qui ne sont traitées par aucune fabrique, il générera un warning du style : Annotation types without processors. Notamment pour les annotation standards de l'API Java. Pour éviter ce "problème", il suffit d'ajouter dans le fichier META−INF/services/com.sun.mirror.apt.AnnotationProcessorFactory, le nom d'une fabrique qui ne fait rien pour ces annotations. Par exemple :

```
public class StandardAnnotationProcessorFactory implements AnnotationProcessorFactory {
  /** Collection contenant le nom des Annotations supportées. */
  protected Collection<String> supportedAnnotationTypes =Arrays.asList(
     // Annotation Standard
      "java.lang.*",
     // Meta−Annotation
     "java.lang.annotation.*");

  /** Retourne la liste des annotations supportées par cette Factory. */
  public Collection<String> supportedAnnotationTypes() {return supportedAnnotationTypes;}

 /** Retourne la liste des options supportées par cette Factory. */
  public Collection<String> supportedOptions() {return Collections.emptyList();}

 /** Retourne AnnotationProcessors.NO_OP (Pas de traitement). */

 public AnnotationProcessor getProcessorFor(Set<AnnotationTypeDeclaration> atds, AnnotationProcessorEnvironment env) {return AnnotationP

}
```

2 SimpleAnnotationProcessor.java

```
public class SimpleAnnotationProcessor implements AnnotationProcessor {
 /** L'environnement du processus d'annotation. */
```

```
 protected final AnnotationProcessorEnvironment env;

 /** Constructeur.
     @param env L'environnement du processeur d'annotation. */
 public SimpleAnnotationProcessor (AnnotationProcessorEnvironment env) {this.env = env;}

 /** Traitement des fichiers sources. */
 public void process() {
  // Instanciation du Visitor
  TODOVisitor todoVisitor = new TODOVisitor(env);
  // On boucle sur toutes les Annotations :
  for (Declaration d: env.getTypeDeclarations()) {
    // On "visite" chacune des déclarations trouvées :
     d.accept(DeclarationVisitors.getSourceOrderDeclarationScanner(todoVisitor,DeclarationVisitors.NO_OP));
  }
 }
}
```

3 TODOVisitor.java

```
public class TODOVisitor extends SimpleDeclarationVisitor {
 protected final AnnotationProcessorEnvironment env;

 public TODOVisitor (AnnotationProcessorEnvironment env) {this.env = env;}

 /** Pour tout type de déclaration, on affiche un message si l'Annotation @TODO est présente...
     De même, on affiche un message pour tous les @TODO contenu dans l'annotation @TODOs */
  @Override
  public void visitDeclaration(Declaration decl) {
   // On regarde si la déclaration possède une annotation TODO
   TODO todo = decl.getAnnotation(TODO.class);
   // Et on l'affiche eventuellement :
   if (todo!=null) printMessage(decl, todo);
   // On fait la même chose pour l'annotation TODOs :
   TODOs todos = decl.getAnnotation(TODOs.class);
   if (todos!=null) {
     // On affiche les message pour tout les TODOs :
     for (TODO t: todos.value()) printMessage(decl,t);
   }
 }
 /** Affiche dans la console l'annotation TODO.
   * @param decl
   * @param todo
   */
  public void printMessage (Declaration decl, TODO todo) {
     m.printNotice(decl.getSimpleName() + " : " + todo.value());
   }
```

4 TODOVisitor.java (2)

On peut également utiliser l'option −Arelease pour encore augmenter l'importance de ces messages lorsqu'on souhaite compiler une version stable de notre application.

```
public class TODOVisitor extends SimpleDeclarationVisitor {
 protected final AnnotationProcessorEnvironment env;

 /** Indique si l'option −Arelease fait partie de la ligne de commande. */
 protected final boolean isRelease;

 public TODOVisitor (AnnotationProcessorEnvironment env) {
  this.env = env;
  isRelease = env.getOptions().containsKey(SimpleAnnotationProcessorFactory.RELEASE);}

 /** Pour tout type de déclaration, on affiche un message si l'Annotation @TODO est présente...
 De même, on affiche un message pour tous les @TODO contenu dans l'annotation @TODOs */
 @Override
 public void visitDeclaration(Declaration decl) {
   // On regarde si la déclaration possède une annotation TODO
```

```
    TODO todo = decl.getAnnotation(TODO.class);
    // Et on l'affiche eventuellement :
    if (todo!=null) printMessage(decl, todo);
    // On fait la même chose pour l'annotation TODOs :
    TODOs todos = decl.getAnnotation(TODOs.class);
    if (todos!=null) {
      // On affiche les message pour tout les TODOs :
      for ( TODO t: todos.value()) printMessage(decl,t);
    }
  }

  /** Affiche dans la console l'annotation TODO.
    * @param decl
    * @param todo */
  public void printMessage (Declaration decl, TODO todo) {
    if (isRelease)
     printReleaseError (decl, todo);
    else
     printDebugingInfo (decl, todo);
   }

  /**
    * Affiche le message pendant la phase de developpement.
    * Les messages IMPORTANT sont affichées comme des 'warnings'.
    * Les messages NORMAL sont affichées comme des 'notes'.
    * Les messages MINEUR sont affichées comme des 'notes' simples (sans position dans le
code).
    * @param decl La déclaration qui contient l'annotation.
    * @param todo L'Annotation affichée.
    */
  public void printDebugingInfo (Declaration decl, TODO todo) {
   Messager m = env.getMessager();
   switch (todo.level()) {
    case IMPORTANT: m.printWarning(decl.getPosition(),decl.getSimpleName() + "␣:␣" + todo.value()); break;

    case NORMAL: m.printNotice(decl.getPosition(),decl.getSimpleName() + "␣:␣" + todo.value() ); break;

    case MINEUR:m.printNotice(decl.getSimpleName() + "␣:␣" + todo.value() );break;
    }
  }

  /**
    * Affiche le message pendant la compilation en mode release.
    * Les messages IMPORTANT sont affichées comme des 'erreurs'.
    * Les messages NORMAL sont affichées comme des 'warning'.
    * Les messages MINEUR sont affichées comme des 'notes'.
    * @param decl La déclaration qui contient l'annotation.
    * @param todo L'Annotation affichée.
    */

   public void printReleaseError (Declaration decl, TODO todo) {
    Messager m = env.getMessager();
    switch (todo.level()) {

      case IMPORTANT: m.printError(decl.getPosition(),decl.getSimpleName() + "␣:␣" + todo.value() );break;

      case NORMAL:m.printWarning(decl.getPosition(),decl.getSimpleName() + "␣:␣" + todo.value() );break;

      case MINEUR:m.printNotice(decl.getPosition(),decl.getSimpleName() + "␣:␣" + todo.value() );break;
    }
   }
}
```

5 Constructeur vides

```
public class EmptyConstructorVisitor extends SimpleDeclarationVisitor {
  /** L'envirronnement du processeur d'annotation. */
  protected final AnnotationProcessorEnvironment env;
```

```
/** Constructeur. @param env L'envirronnement du processeur d'annotation. */
public EmptyConstructorVisitor (AnnotationProcessorEnvironment env) {this.env = env;}

/** Méthode appelé pour chaque déclaration d'une classe. */

@Override
public void visitClassDeclaration(ClassDeclaration classDecl) {
 // Si on n'a pas de constructeur vide et que l'annotation EmptyConstructor a été trouvé
 // --> On affiche un message d'erreur :
 if (!hasEmptyConstructor(classDecl) && hasAnnotation(classDecl,EmptyConstructor.class))
 {
  env.getMessager().printError( classDecl.getPosition(),"Un constructeur vide est requis par @EmptyConstructor.");
 }
}

/** Cette méthode indique si la déclaration de classe passée en paramètre possède unn constructeur vide.
 * @param classDecl La déclaration de classe à analyser.
 * @return true si la classe possède un constructeur vide, false sinon. */
public boolean hasEmptyConstructor (ClassDeclaration classDecl) {
   // On parcourt la liste des constructeurs :
   for (ConstructorDeclaration c: classDecl.getConstructors() ) {
    if (c.getParameters().isEmpty()) return true; // On a trouvé un constructeur vide
   }
 return false; // Pas de constructeur vide
}
```