Exploiter les annotations à la compilation

Objectifs

— Définir des annotations

— Définir un processeur d'annotation

Exercice 1 : Vérifier les classes utilitaires

On souhaite pouvoir déclarer une classe comme « utilitaire » en définissant une annotation `Utility`. Une classe utilitaire est une classe qui ne contient que des méthodes (plus généralement des membres) de classe, donc déclarées **`static`**. De plus cette classe ne devrait pas être héritée [1] (elle doit être déclarée **`final`**) et ne doit pas être instanciée (elle ne doit définir que le constructeur par défaut qui doit être privé). Les classes java.lang.Math, java.util.Collections ou java.lang.System sont des exemples de classes utilitaires.

1.1. Définir l'annotation `Utility`.

La classe UtilityClass.java doit compiler sans erreur. La classe WrongUseOfUtility ne doit pas compiler (plusieurs erreurs).

1.2. Lire la classe `UtilityProcessor` du listing 1 et retrouver les éléments décrits ci-après.

La classe `UtilityProcessor` est un processeur d'annotation. Il est exécuté par le compilateur lors de la compilation du code source Java et peut signaler au compilateur des messages d'information, d'avertissement ou d'erreur.

Cette classe spécialise la classe abstraite AbstractProcessor qui réalise l'interface Processor. Cette classe abstraite définit un attribut protégé `processingEnv` qui donne accès à l'environnement, en particulier à l'objet de type Messager qui permet de signaler des messages au compilateur.

Quand on définit un processeur d'annotation, on doit préciser les annotations qu'il sait traiter et avec quelle version d'un source Java il fonctionne. Ceci pourrait être fait en (re)définissant les méthodes de `AbstractProcessor` mais c'est plus lisible et pratique d'utiliser des annotations. C'est la raison d'être des deux annotations `SupportedAnnotationTypes` et `SupportedSourceVersion`.

Il ne reste alors plus qu'à définir la méthode abstraite `process`. Elle prend en paramètre la liste des annotations à traiter et un deuxième objet qui permet d'avoir accès à l'environnement de compilation (`roundingEnvironment`), en particulier aux éléments de l'arbre abstrait. Par exemple, l'environnement de compilation permet de retrouver les élément marqués avec l'annotation (`getElementsAnnotatedWith`).

1.3. Exécuter le processeur d'annotation pour vérifier que tout semble fonctionner correctement. Dans un terminal, on compile d'abord notre processeur d'annotation puis on utilise l'option `-processor` du compilateur pour compiler les autres classes :

1. Une pratique courante était d'hériter d'une classe utilitaire pour pouvoir utiliser directement ces méthodes sans avoir à les préfixer du nom de la classe. L'introduction de **`import static`** depuis Java5 a rendu inutile cette mauvaise utilisation de l'héritage.

```java
import javax.annotation.processing.*;
import javax.lang.model.SourceVersion;
import javax.lang.model.element.*;
import java.util.*;
import javax.tools.Diagnostic.Kind;

/** Check that a class marked {@code @Utility} is indeed a utility class. */
@SupportedAnnotationTypes("Utility")
@SupportedSourceVersion(SourceVersion.RELEASE_11)
public class UtilityProcessor extends AbstractProcessor {

    @Override
    public boolean process(
            Set<? extends TypeElement> annotations,
            RoundEnvironment roundingEnvironment)
    {
        Messager messager = processingEnv.getMessager();
        messager.printMessage(Kind.NOTE,
                "UtilityProcessor executed.");
        messager.printMessage(Kind.WARNING,
                "The provided UtilityProcessor class is wrong.  Correct it!");
        for (TypeElement te : annotations) {
            for (Element elt : roundingEnvironment.getElementsAnnotatedWith(te)) {
                if (false) {    // elt is a class
                    // Check that the class is declared final

                    // Check that enclosed elements are static

                } else {
                    messager.printMessage(Kind.ERROR,
                            "@Utility applies to class only:", elt);
                }
            }
        }
        return true;
    }

}
```

FIGURE 1 – La classe `UtilityProcessor`

```
javac UtilityProcessor.java
javac -processor UtilityProcessor UtilityClass.java
...
```

1.4. Compléter la classe `UtilityProcessor` pour que la compilation de `UtilityClass` ne provoque plus d'erreur. L'interface `UtilityInterface` doit provoquer des erreurs (une interface ne peut pas être utilitaire). La classe `WrongUtilityClass` doit provoquer des erreurs.

On consultera la documentation de l'interface Element, en particulier les méthodes `getKind()`, `getEnclosedElements()`, `getModifiers()`...

1.5. Dans la version précédente, nous avons été obligé de faire des tests sur le type des éléments (au travers de `getKind()`) et de transtyper les éléments pour avoir accès aux méthodes spécifiques. L'API fournit un visiteur qui évite ces tests de type et transtypages : la méthode `accept` sur `Element` accepte un objet du type ElementVisitor. L'interface ElementVisitor a plusieurs réalisations suivant la version du langage et le type de traitement que l'on souhaite faire. Nous allons utiliser `ElementKindVisitor8` qui permet de visiter les éléments en fonction de leur nature.

Écrire la classe UtilityProcessor en définissant un visiteur de type `ElementKindVisitor8`.