TP1 Java

Y. ALJ

16 mars 2020

1 Préliminaire

Séparation des sources et des classes compilées Nous allons organiser nos fichiers de sorte que le code source des classes (fichiers .java) soit séparés des fichiers contenant le bytecode (fichier .class).
Afin que les fichiers créés pendant la compilation soient dans un autre répertoire on utilise l'option `-d`.
Exemple :

```
javac -d monRepertoireDeTravail/JAVA/classes/ HelloWorld.java
```

Pour l'exécution on utilise alors l'option `-classpath`.
Exemple :

```
java -classpath monRepertoireDeTravail/JAVA/classes/ HelloWorld
```

2 Exercices

Exercice 1

Partie 1 :

1. Créer un répertoire `tp1` (avec la commande `md` sous Windows, `mkdir` sous Linux ou avec l'explorateur de votre choix).
   Nous allons mettre les fichiers sources .java dans un sous répertoire de `tp1` nommé `src`. Le résultat de la compilation (.class) devrait se trouver dans un sous répertoire de `tp1` appelé `bin`.
2. Avec Notepad++ créer un nouveau fichier nommé `HelloWorld.java` dans le répertoire `tp1/src/`.
3. Écrire une classe appelée `HelloWorld` qui affiche "bonjour tout le monde" à la sortie standard.
4. On se place avec la commande `cd` dans le répertoire `tp1` et on compile le fichier `HelloWorld.java` avec l'option `-d` (voir préliminaire).
5. Exécuter avec la commande `java` du préliminaire.

Partie 2 :

1. Effacer le contenu de la fonction main (uniquement ce qui se trouve à l'intérieur des accolades).
2. Créer un nouvel attribut privé de nom `message` de type `String`.
3. Ajouter à votre classe `HelloWorld` deux méthodes :
   — Un constructeur qui attribue une valeur au message.
   — Une méthode `public String getMessage()` qui renvoie la valeur du message.
4. Compiler à nouveau avec l'option `-d`.
5. Exécuter. Quelle remarque peut on faire ? Pourquoi ?
6. Dans la fonction `main` Créer une instance de la classe `HelloWorld` et afficher son message. On utilisera pour cela la fonction `System.out.println(String t)`.

3 Package

3.1 Introduction

Un package est un regroupement de plusieurs classes selon un thème précis. Au niveau du code source, un package n'a pas d'existence explicite, il n'est réellement créé que dès que sa première classe est compilée. Pour indiquer qu'une classe appartient à un package on utilise au tout début du fichier contenant le code source le mot-clé package . Pour assigner la classe Livre au package bibliotheque, on écrira :

```
1 package bibliotheque;
2 public class Livre {
3    //contenu de la classe
4 }
```

Un package peut être imbriqué dans un package plus général et ainsi de suite (comme des dossiers). Pour désigner un package document imbriqué dans le package bibliotheque on écrira par exemple :

```
1 package bibliotheque.document;
```

A la compilation, des répertoires sont créés pour correspondre aux packages des classes compilées. Les fichiers contenant le bytecode sont enregistrés dans le répertoire de leur package. Une classe est alors désignée par son nom de package, suivie d'un point puis du nom de la classe. Par exemple, pour exécuter la classe Livre on utilise l'instruction : `java bibliotheque.Livre`

Pour que la référence à la classe Livre soit effective, il faut que le répertoire contenant son package soit accessible par la machine virtuelle. Il y a deux solutions pour cela :

1. Exécuter la classe à partir du répertoire contenant le répertoire bibliotheque. Dans ce cas là, le package bibliotheque et les classes qu'il contient sont directement trouvés car ils sont dans le répertoire d'exécution.
2. Exécuter la classe à partir d'un autre répertoire. Il faut alors préciser le chemin d'accès du répertoire contenant le package bibliotheque. Cela se fait en utilisant l'option -classpath comme précédemment :
   `java –classpath /MonRepertoire/Java/classes bibliotheque.Livre`

3.2 Importation de package

Quand une classe fait référence à une autre classe, elle doit l'importer sauf si elles se trouvent dans le même package. Pour cela le mot-clé import est utilisé au début du fichier source, un peu comme include de C, comme dans l'exemple qui suit :

```
1 package client;
2
3 import bibliotheque.Livre;
4
5 public class Lecteur {
6    Livre[] emprunte;
7    ...
8 }
```

De cette manière la classe Lecteur peut utiliser la classe Livre. Il est également possible d'importer toutes les classes d'un package en utilisant le symbole * comme suit :

import bibliotheque.*;

Exercice : Créer un package `ma.emsig.helloworld` (donc un répertoire ma/emsig/helloworld dans le répertoire src) et déplacez-y votre fichier HelloWorld.java. Déclarez que la classe HelloWorld appartient au package ma.emsig.helloworld.

1. Compiler avec la commande suivante :
   `javac –d bin src/ma/emsig/helloworld/HelloWorld.java`
2. Exécuter avec la commande suivante :
   `java –classpath bin ma.emsig.helloworld.HelloWorld`

4 La documentation avec javadoc

javadoc est un autre programme du SDK qui créé une documentation automatique à partir du code source de classes Java. Cette documentation automatique décrit les membres d'une classe dans un format HTML. Pour créer tous les fichiers de documentation d'un package, on utilise javadoc comme suit :
`javadoc -d <répertoire de destination> <nom des packages>` Le fichier principal de documentation est index.html créé dans le répertoire de destination.
Remarque : par défaut la commande javadoc ne génère la documentation que pour les classes et membres public, donc pensez à établir la visibilité de votre classe à public. Vous pouvez également spécifier à javadoc que vous voulez également générer la documentation pour les membres privés avec l'option -private :
`javadoc -d <répertoire de destination> -private <nom des packages>`

Générez la documentation de votre package dans un répertoire tp1/doc.

Quelques tags javadoc javadoc peut interpréter des commentaires spécifiques introduits dans le code source pour enrichir la documentation générée. Ces commentaires se situent juste avant la déclaration d'une classe, d'un attribut ou d'une méthode. Ils commencent par /** et se terminent par */. Ces commentaires contiennent une partie textuelle libre et des tags interprétés pour certains commentaires spécifiques.

Quelques tags fréquemment utilisés
— @author : l'auteur d'une classe
— @version : le numero de version d'une classe
— @see : une référence à une classe ou un membre d'une classe intéressant pour la classe ou la méthode commentée
— @param x : Une description du paramètre d'entrée x d'une méthode
— @return : Une description de la valeur renvoyée par une méthode

Exemple de classe commentée pour javadoc

```
1  package bibliotheque;
2  /**
3   * Cette classe est utilisee pour representer un livre.
4   *
5   * @author Laurent Vercouter
6   */
7  public class Livre extends Produit {
8
9      /**
10     * Le titre du livre
11     */
12     private String titre;
13
14     /**
15     * L auteur du livre
16     */
17     private String auteur;
18
19     /**
20     * Le constructeur de la classe Livre
21     *
22     * @param tit Le titre du livre
23     * @param aut L auteur du livre
24     */
25     public Livre(String tit, String aut) {
26         titre = tit;
27         auteur = aut;
28     }
29
30     /**
31     * Cette methode renvoie une chaene de caracteres qui decrit
32     * textuellement le livre (par son titre, son auteur et lediteur)
33     *
34     * @return Une chaine de caractere decrivant le livre
35     */
36     public String description() {
```

```
37          return "\""+titre+"\" de "+auteur+" edite par "+editeur;
38      }
39  }
```

Exercice : Ajoutez des commentaires et des tags pour que vos classes soient correctement documentées. Par la suite, toutes vos classes devront être commentées de cette manière.