TP01 : Prise en main

Objectifs

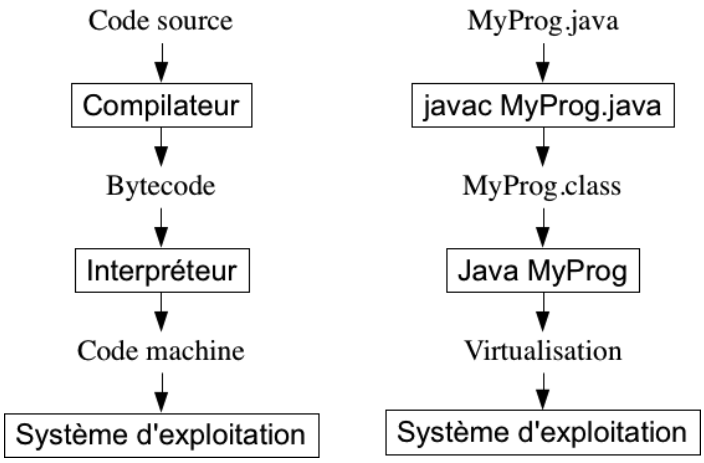
- Savoir compiler un code Java
- Savoir documenter un code Java
- Savoir exécuter un code Java
- Savoir lire les messages d'erreur Java
- Savoir parcourir et utiliser l'API du SDK

Présentation générale de Java et du Standard Development Kit (SDK)

Le SDK regroupe un ensemble d'outils permettant principalement de compiler et d'exécuter des programmes Java. Actuellement, la dernière version stable est Java SE 1.7.0 et est téléchargeable sur le site Java SE d'Oracle.

Java est un langage interprété, ce qui signifie qu'un programme compilé n'est pas directement exécutable par le système d'exploitation mais il doit être interprété par un autre programme, qu'on appelle interpréteur.

Exemple :



Le compilateur javac

javac est le compilateur fourni dans le SDK permettant de compiler un fichier source .java. Ce code source est alors compilé par le compilateur javac en un langage appelé *bytecode* et enregistre le résultat dans un fichier dont l'extension est `.class`. Pour compiler une classe `Livre` contenue dans un fichier `Livre.java` on exécutera donc l'instruction suivante:

```
javac Livre.java
```

L'interpréteur java et la machine virtuelle

Le *bytecode* n'est pas directement exécutable. Il doit être interprété par une **machine virtuelle** Java qui le traduit en un langage adapté au système d'exploitation. Le programme java fourni dans le SDK lance une machine virtuelle pour interpréter une classe compilée. **Attention :** l'extension .class ne doit pas être précisée dans l'instruction d'interprétation. Pour exécuter la classe `Livre` compilée dans un fichier `Livre.class`, on écrit donc : **`java`** `Livre`

Séparation des sources et des classes compilées

En général, les fichiers contenant le code source des classes sont séparés des fichiers contenant le *bytecode* pour mieux structurer les différentes parties du programme. Pour que les fichiers créés par une compilation le soient dans un autre répertoire on utilise l'option **`-d`**.

Par exemple : `javac -d /MonRepertoire/Java/classes Livre.java`

Lorsque le *bytecode* ne se situe pas dans le répertoire courant, il faut signaler à la commande java où il se trouve grâce à l'option `classpath`.

Par exemple : `java -classpath /MonRepertoire/Java/classes Livre`

Exercice 1

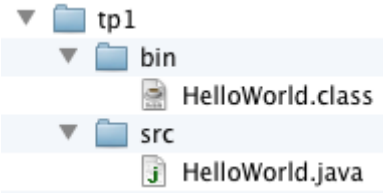
Créez un répertoire tp1 (avec la commande ***md*** sous Windows, ***mkdir*** sous Linux ou avec l'exploreur de votre choix). Nous allons mettre les sources dans un répertoire nommé src et les `.class` dans un répertoire nommé `bin` (à créer comme précédemment). Ouvrez un éditeur de texte, puis créez un fichier vide nommé `HelloWorld.java` dans le répertoire `tp1/src`.

Exercice 2

Ecrivez le code source d'une classe `HelloWorld` dans le fichier `HelloWorld.java`. La classe `HelloWorld` contient un attribut **privé** `message` qui est du type `String`. `HelloWorld` doit également contenir deux méthodes : un constructeur qui attribue une valeur au message et une méthode `public String getMessage()` qui renvoie la valeur du message. Compilez cette classe en séparant bien le code source et le *bytecode* grâce à la commande suivante, **depuis le répertoire** `tp1` :

```
javac -d bin src/HelloWorld.java
```

Ceci signifie que la classe à compilée est dans le fichier `HelloWorld.java` qui est dans le répertoire `src` et que le fichier `.class` doit être généré dans le répertoire `bin`, comme dans la figure suivante :



Classe exécutable et méthode `main()`

La classe `HelloWorld` n'est pas exécutable dans son état actuel. Une classe exécutable est une classe qui contient une méthode spécifique (`main`) utilisée comme point de départ de l'exécution. La méthode `main` d'une classe s'écrit de la manière suivante :

```
public static void main(String[] args){
   //début du code à exécuter
}
```

Exercice 3

Ajoutez une méthode `main` à la classe `HelloWorld`. Dans cette méthode créez une instance de `HelloWorld` et affichez son message. Pour afficher du texte à l'écran utilisez la méthode `System.out.println(String t)`. Compilez comme précédemment.

Savoir lire les messages d'erreur Java

En cas d'erreur, le compilateur affiche notamment ce qu'on appelle une **trace de la pile d'appels** qui ont amené à cette erreur. Cette liste d'appels est affichée de l'appel le plus interne à l'appel le plus externe. Pour trouver l'origine de l'erreur (si elle est due à votre code), parcourez cette liste jusqu'à trouver une classe de votre propre code. Le message d'erreur Java affiche également la ligne du code de l'appel. Pratique pour directement aller à la source de l'erreur. De plus, Java faisant usage de la notion d'exception, en lisant le type d'exception qui a généré l'erreur il est également possible d'en déduire l'origine. Par exemple, si `NullPointerException` est affichée, il y a de forte chance que vous ayez oublié d'initialiser un objet (par appel à `new`) avant de faire appel à un de ses attributs ou méthodes.

Exercice 4

Exécutez la classe `HelloWorld` grâce à la commande suivante :

```
java -classpath bin HelloWorld
```

car le fichier `.class` à exécuter ne se situe pas dans le répertoire courant.

Les arguments de la méthode main

Vous avez vu que la méthode `main` possédait la signature suivante :

```
public static void main(String [] args) {
   ...
}
```

Le tableau `args` contient tous les arguments passés en paramètres du programme Java. Par exemple, suite à la ligne de commande

```
java MonProgramme texte1 127 unfichier.txt
```

le tableau `args` contiendra les élément suivants :

- `args[0]` : la chaîne de caractères `"texte1"`
- `args[1]` : la chaîne de caractères `"127"`
- `args[2]` : la chaîne de caractères `"unfichier.txt"`

Packages

Notion de package et de classpath

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 `biliotheque`, on écrira :

```
package bibliotheque;
public class Livre {
   //contenu de la classe
}
```

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 :

```
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
```

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 :

```
package client;

import bibliotheque.Livre;

public class Lecteur {
   Livre[] emprunte;
   ...
}
```

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 5

Créez un package `sn.uasz.helloworld` (donc un `sn/uasz/helloworld` dans le répertoire src) et déplacez-y votre fichier `HelloWorld.java`. Déclarez que la classe `HelloWorld` appartient au package `sn.uasz.helloworld`.
Compilez avec la commande suivante :
`javac -d bin src/sn/uasz/helloworld/HelloWorld.java`
Puis exécutez-la avec la commande suivante :
`java -classpath bin sn.uasz.helloworld.HelloWorld`
Vous devriez obtenir l'arborescence suivante : `bin -> sn -> uasz -> helloworld.`