Quelques bases du langage

Plan

Commentaires

Une seule ligne : //

```
int i = 0;   // Initialize the loop variable
```

Plusieurs lignes : /* infos */

```
/*
 * First, establish a connection to the server.
 * If the connection attempt fails, quit right away.
 */
```

Commentaires pour la javadoc : /** infos */

```
/**
 * Upload a file to a web server.
 *
 * @param file The file to upload.
 * @return <tt>true</tt> on success,
 *         <tt>false</tt> on failure.
 * @author David Flanagan
 */
```

Mots réservés

- tous les détails sur Wikipédia

Convention pour les identifiants

Pas d'accents dans le code !

Par convention :

- Le nom des classes commencent toujours par une majuscule : e.g. **Bidule**
  $\Rightarrow$ **donc les fichiers aussi !**
- le nom d'une **variable**, d'un **attribut** ou d'une **méthode** commence par une **minuscule** : e.g. **int time ;**
- pour les noms de plus d'un mot : on utilise le ***Camel case*** :
  - exemples pour une classe : **MaClasse** , **MonPremierObjet**
  - exemples pour un attribut ou une méthode : **monAgeActuel** , **maPremiereMethode**

Les littéraux

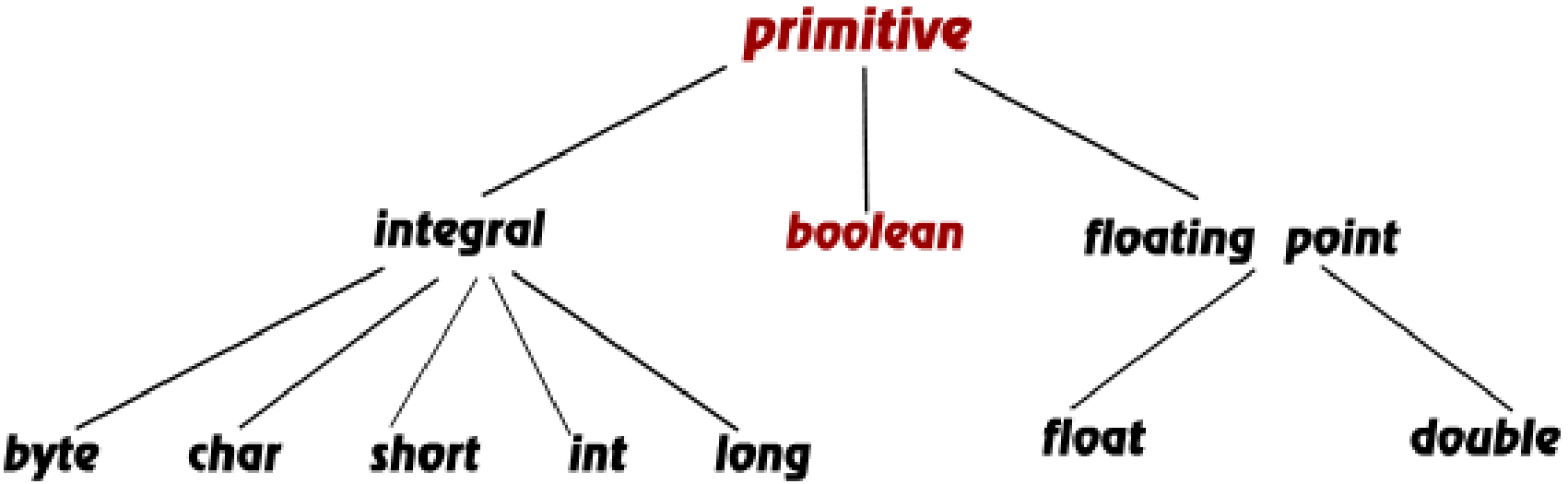
Les littéraux sont des valeurs qui peuvent apparaitrent directement dans le code Java.

Cela inclut :

- les entiers naturels et les flottants : 1 1.0 1.2
- les caractères : 'a' '2'
- les chaînes de caractères : "a" "the one" "24"
- les mots réservés **true false** (les valeurs booléennes) et **null** (l'objet null)

Types primitifs

▸ tous les détails



Remarque

Tout le reste est "objet" en Java !

Le classpath : qu'est-ce que c'est ?

- C'est une **variable d'environnement** (OS) utilisée par la JVM.
- Littéralement : « le chemin des classes » : les endroits du système de fichiers où se trouvent les classes.
- Plus exactement : les **"endroits" où débutent des arborescences de packages** (cf. section suivante)
- **Elle pointe sur** :
  - des répertoires où se trouvent **des arborescences de fichiers *.class***
  - et/ou **des fichiers « *.jar* »** (Java Archive : compressée)
- Utilisée par **java** et **javac** pour trouver les fichiers nécessaires (dépendances) à la compilation et à l'exécution (fichiers *.class* ou *.jar*).

Spécifier le classpath est donc nécessaire pour :

- charger les librairies utilisées pour la compilation et l'exécution

Spécification explicite du classpath

Exemple avec un **fichier source dans** */home/joe/java/src/*

Première solution : l'option -cp (-classpath) de javac et java

```
joe@iut:~$ javac /home/joe/java/src/HelloWorldApp.java
joe@iut:~$ java -cp /home/joe/java/src/ HelloWorldApp
Hello World!
joe@iut:~$
```

-cp permet à javac de trouver les dépendances, pas les sources !

javac n'utilise pas le classpath pour trouver les sources cibles :

```
joe@iut:~$ javac -cp /home/joe/java/src/ HelloWorldApp.java
javac: file not found: HelloWorldApp.java
Usage: javac <options> <source files>
use -help for a list of possible options

joe@iut:~$ javac -sourcepath /home/joe/java/src HelloWorldApp.java
```

Spécification explicite du classpath

Deuxième solution (**À ÉVITER**) : régler définitivement le classpath en faisant une variable d'environnement du système d'exploitation

Soit pour la session courante (la console)

- Windows : SET CLASSPATH = . ;c :\ mesprog
- Linux : CLASSPATH=. :~/mesprog

Soit définitivement (comme pour le path)

- Windows : propriétés du poste de travail -> avancé -> variables d'environnement (nouveau) : CLASSPATH=. ;c :\ mesprog
- Linux : dans le fichier .bashrc de votre racine :
  CLASSPTH=. :~/mesprog
  export CLASSPATH

Classpath par défaut

Comment java et javac trouvent-ils les classes de base ? !

Pourquoi n'y-a-t-il pas besoin de régler le classpath pour le répertoire courant ? !

Plusieurs variables ont des valeurs par défaut au démarrage d'une JVM

- *sun.boot.class.path* : **JDKInstalDir. . . /jre/lib/rt.jar** : ce fichier contient toutes les classes de base, compilées et classées suivant leur package (sans ce fichier : plus rien !).
- *java.class.path* : **"."** (répertoire courant). Le classpath contient par défaut le répertoire dans lequel la commande a été lancée (le *working dir*).

Remarque : les sources du langage sont aussi disponibles :

Package

Définition

- Un package est un regroupement de classes conceptuellement cohérent par rapport à leurs fonctionnalités (ex : *java.io java.util*)
- Un package définit ainsi un espace de nommage (permet d'éviter les conflits de nom).

Les packages en Java

- Plus de 4000 classes sont directement accessibles dans le JDK ! !
- Elles sont classées dans des répertoires suivant leurs fonctionnalités.
- Ces **répertoires** correspondent exactement aux **packages**
- $\rightarrow$ **Le nom d'un package correspond exactement au répertoire où se trouvent ses classes**.

Package

Utilisation d'une classe contenue dans un package

- Pour pouvoir utiliser une classe il faut obligatoirement l'importer explicitement en utilisant le mot clé *import* en début de fichier.

```
import java.util.ArrayList;
```

Pour que cela fonctionne, il faut donc que :

le **classpath** pointe sur un répertoire contenant un répertoire *java* contenant lui-même un répertoire *util* dans lequel se trouve un fichier *ArrayList.class*

Package

Pour utiliser une classe, il faut absolument l'avoir importée !

Question ?

- Alors, pourquoi peut-on utiliser la classe *System* (e.g. *System.out.println*) sans aucun « import » ? !

Package

Réponse

- tout le package **java.lang est importé par défaut** et en particulier **Object** !

```
public class MonProg{
```

```
public class MonProg extends Object{
```

Créer ses propres packages

Règle d'or en Java

- toute nouvelle classe doit impérativement se trouver dans un package !
- utilisation du mot clé *package* en début de fichier (avant les imports)

```
package prog;

public class MonProg{

public static void main(String[] args)
{
  System.out.println("hello");
}

}
```

Attention !

- le fichier *MonProg.java* doit se trouver **dans le répertoire prog**

Compilation et exécution

Dans un terminal

```
joe@iut:~$ pwd
/home/joe/mesprog
joe@iut:~$ cd prog
joe@iut:~$ javac MonProg.java
joe@iut:~$ java MonProg
```

Compilation et exécution

Dans un terminal

```
joe@iut:~$ pwd
/home/joe/mesprog
joe@iut:~$ cd prog
joe@iut:~$ javac MonProg.java
joe@iut:~$ java MonProg
```

Question ?

- Cela va-t-il marcher ?

Compilation et exécution

Dans un terminal

```
joe@iut:~$ pwd
/home/joe/mesprog
joe@iut:~$ cd prog
joe@iut:~$ javac MonProg.java
joe@iut:~$ java MonProg
```

Réponse : non !

Exception in thread "main"
java.lang.NoClassDefFoundError :
MonProg (wrong name : prog/MonProg)

Compilation et exécution

Ne pas oublier le principe du classpath

- Il faut que le classpath soit cohérent pour pouvoir exécuter !

Dans un terminal

```
joe@iut:~$ pwd
/home/joe/mesprog
joe@iut:~$ javac prog/MonProg.java
joe@iut:~$ java prog/MonProg
hello !
joe@iut:~$ java prog.MonProg
hello !
```

Utiliser ses propres packages

Pour utiliser ses propres packages depuis un autre package, il faut ajouter le répertoire contenant le package désiré au classpath :

```
package tp1;

import prog.MonProg;

public class NouveauProg{

public static void main(String[] args)
{
  System.out.println("nouveau");
  MonProg.main(null);
}

}
```

Pour compiler (depuis le répertoire au dessus de tp1) :

- javac -cp ~/mesprog tp1/NouveauProg.java

Utiliser ses propres packages

```
package tp1;

import prog.MonProg;

public class NouveauProg{

public static void main(String[] args)
{
  System.out.println("nouveau");
  MonProg.main(null);
}

}
```

Pour exécuter (depuis le répertoire au dessus de tp1) :

- > java **-cp** . :~/**mesprog** tp1.NouveauProg
  > nouveau
  > hello
- Note : il faut inclure "." (répertoire courant) dans le cp car en modifiant sa valeur, on écrase celle par défaut.

Utiliser ses propres packages

```
package tp1;

import prog.MonProg;

public class NouveauProg{

public static void main(String[] args)
{
   System.out.println("nouveau");
   MonProg.main(null);
}

}
```

Pour exécuter (depuis n'importe où) :

- > java **-cp /cheminQuiMèneÀtp1 :~/mesprog** tp1.NouveauProg
  > nouveau
  > hello

Résumé global

Le classpath

- La variable classpath permet de spécifier où se trouvent des packages.
- Elle peut contenir plusieurs entrées
- Il est indispensable de la spécifier pour que le programme marche en toute circonstance.

Les packages

- Les classes du langage Java sont organisées en packages (rt.jar).
- le mot clé *package* permet de spécifier le package d'appartenance d'une classe.
- Un bon programme est obligatoirement organisé en packages (structuration logique des classes).