Votre premier programme Java (travail à réaliser)

Table des matières

1 Les différentes étapes de création d'un programme Java

1.1 Création des fichiers source

Dans un premier temps, il vous faut créer un ou plusieurs fichiers de code source, selon l'importance de votre programme.
Tout code java est contenu à l'intérieur d'une classe qui est elle-même contenue dans un fichier portant l'extension `java`.
Plusieurs classes peuvent exister dans un même fichier .java mais une seule peut être déclarée publique, et c'est cette classe qui donne son nom au fichier.
Comme beaucoup d'autres langages de programmation, les fichiers source Java sont des fichiers de texte sans mise en forme.
Un simple éditeur de texte capable d'enregistrer au format texte ASCII, tel que le Bloc-notes de Windows ou VI sous Unix, est suffisant pour écrire des sources Java.
Une fois le code de votre fichier source écrit, il faut l'enregistrer avec l'extension `java` qui est l'extension des fichiers source.
Si vous utilisez le Bloc-notes de Windows, faites attention lors de l'enregistrement de votre fichier que le Bloc-notes n'ajoute pas une extension `.txt` au nom de votre fichier. Pour éviter ce problème, nommez votre fichier avec l'extension `java`, le tout placé entre guillemets.
II y a toutefois mieux qu'un simple éditeur de texte pour développer. Vous pouvez, moyennant le coût d'une licence, utiliser des outils commerciaux ou encore mieux utiliser des produits open source comme l'excellent **Eclipse**. Il s'agit au départ d'un projet IBM mais de nombreuses sociétés se sont jointes à ce projet (Borland, Oracle, Merant...). C'est un outil de développement Java excellent et gratuit auquel on peut lier d'autres applications tiers via un système de plug-in. Sun/Oracle propose également **NetBeans** un outil très efficace et simple d'utilisation. Jet Brain propose **IntelliJ IDEA** .

1.2 Compiler un fichier source

Une fois votre fichier source réalisé et enregistré avec l'extension .java, il faut compiler votre fichier.

Pour compiler un fichier source Java, il faut utiliser l'outil en ligne de commande javac fourni avec le SDK.

- Ouvrez une fenêtre Invite de commandes.
- À l'invite de commandes, placez-vous dans le dossier contenant votre fichier source (.java), à l'aide de la commande **cd** suivie d'un espace puis du nom du dossier qui contient votre fichier source.
- Une fois que vous êtes dans le bon dossier, vous pouvez lancer la compilation de votre fichier à l'aide de la commande suivante à l'invite de commandes :

javac <nomFichier>.java

avec

javac : compilateur Java en ligne de commande, fourni avec le JDK.

<nomFichier> : nom du fichier source Java.

.java : extension qui indique que le fichier est une source Java.

Si vous voulez compiler plusieurs fichiers source en même temps, il suffit de saisir la commande précédente, puis d'ajouter les autres fichiers à compiler en les séparant par un espace.

javac <nomFichierl>.java <nomFichier2>.java

Si au bout de quelques secondes vous voyez apparaître de nouveau l'invite de commandes MS-DOS, c'est que votre fichier ne contient pas d'erreur et qu'il a été compilé. En effet le compilateur n'affiche pas de message quand la compilation se déroule correctement.

Le résultat de la compilation d'un fichier source Java est la création d'un fichier binaire portant le même nom que le fichier source mais avec l'extension .class.

Un fichier binaire .class contient le pseudo-code Java qui peut être interprété par la machine virtuelle Java. Si par contre, vous voyez apparaître une suite de messages dont le dernier vous indique un nombre d'erreurs, c'est que votre fichier source contient des erreurs et que javac n'a donc pas réussi à le compiler.

Dans ce cas, il vous faut corriger votre source.

Pour vous aider à trouver les erreurs de code de votre ou de vos fichiers source, javac vous fournit des informations très utiles :

<nomFichier.java> : <numLigne> : <message> <ligne de code>

avec

<nomFichier>

Nom du fichier source Java qui contient une erreur.

<numLigne>

Numéro de la ligne de votre fichier source où javac a décelé une erreur.

<message>

Message indiquant le type de l'erreur.

<ligne>

Ligne de code contenant une erreur, javac indique par une flèche où est située l'erreur dans la ligne.

Après avoir corrigé votre code, recompilez votre fichier. Si javac vous indique toujours des erreurs, renouvelez l'opération de correction puis de recompilation du fichier jusqu'à obtenir la création du fichier binaire .class.

Par défaut les fichiers compilés sont créés dans le même répertoire que vos fichiers source. Vous pouvez indiquer à l'outil javac de les créer dans un autre répertoire à l'aide de l'option -d "directory".

1.3 Exécuter une application

Une application Java est un programme autonome, semblable aux programmes que vous connaissez, mais qui, pour être exécuté, nécessite l'emploi d'un interpréteur Java (la machine virtuelle Java) qui charge la méthode `main()` de la classe principale de l'application.

Pour lancer l'exécution d'une application Java, il faut utiliser l'outil en ligne de commande `java` fourni avec le JDK.

- Ouvrez une fenêtre **Invite de commandes**. Placez-vous dans le répertoire qui contient le ou les fichiers binaires (.class) de votre application, puis saisissez la commande avec la syntaxe suivante :

  `java <fichierMain> <argumentN> <argumentN+l>`

  `java` : outil en ligne de commande qui lance l'exécution de la machine virtuelle Java.

  `<fichierMain>` : est obligatoirement le nom du fichier binaire (.class) qui contient le point d'entrée de l'application, la méthode `main()`. Important : ne mettez pas l'extension .class après le nom du fichier car ceci est fait implicitement par la machine virtuelle Java.

  `<argumentN> <argumentN+1>` : éventuels arguments de ligne de commande à passer à l'application à l'exécution de celle-ci.

Si vous avez lancé l'exécution correctement (syntaxe correcte, avec le fichier contenant la méthode **`main()`**, vous devez voir apparaître les éventuels messages que vous avez insérés dans votre code. Si par contre vous voyez apparaître un message d'erreur semblable à **`Exception in thread "main" java.lang.NoClassDefFoundError :...`** c'est que votre programme ne peut pas être exécuté.

Plusieurs raisons peuvent en être la cause :

- Le nom du fichier à exécuter ne porte pas le même nom que la classe (différence entre majuscules et minuscules).
- Vous avez saisi l'extension `.class` après le nom du fichier à exécuter sur la ligne de commande.
- Le fichier que vous exécutez ne contient pas de méthode `main()`.
- Vous essayez d'exécuter un fichier binaire (`.class`) qui est situé dans un autre répertoire que celui d'où vous lancez l'exécution.

2 Votre première application Java

2.1 Squelette d'une application

Une application Java est un programme autonome qui peut être exécuté sur n'importe quelle plate-forme disposant d'une machine virtuelle Java.

Tout type d'application peut être développé en Java : interface graphique, accès aux bases de données, applications client/serveur, multithreading...

Une application est composée au minimum d'un fichier .class qui doit lui-même contenir au minimum le point d'entrée de l'application, la méthode `main()`.

Exemple de la structure minimum d'une application

```
public class MonApplication {

      public static void main(String arguments[]) {

        /* corps de la méthode principale */

          System.out.println("Bonjour monde!") ;

      }

}
```

Si l'application est importante, il est possible de créer autant de classes que nécessaire. Les classes qui ne contiennent pas la méthode **`main()`** sont nommées classes auxiliaires.

La méthode **`main()`** est le premier élément appelé par la machine virtuelle Java au lancement de l'application.

Le corps de cette méthode doit contenir les instructions nécessaires pour le lancement de l'application, c'est-à-dire la création d'instances de classe, l'initialisation de variables et l'appel de méthodes.

Idéalement, la méthode **`main()`** peut ne contenir qu'une seule instruction.

La déclaration de la méthode (ou signature) **`main()`** se fait toujours suivant la syntaxe suivante :

`public static void main(String <identificateur>[ ] ) {...}`

`public`

Modificateur d'accès utilisé pour rendre la méthode accessible à l'ensemble des autres classes et objets de l'application, et également pour que l'interpréteur Java puisse y accéder de l'extérieur au lancement de l'application.

`static`

Modificateur d'accès utilisé pour définir la méthode `main()` comme étant une méthode de classe. La machine virtuelle Java peut donc appeler cette méthode sans avoir à créer une instance de la classe dans laquelle elle est définie.

`void`

Mot clé utilisé pour indiquer que la méthode est une procédure qui ne retourne pas de valeur.

`main`

Identificateur de la méthode.

`String <identificateur>[ ] :`

Paramètre de la méthode, c'est un tableau de chaînes de caractères. Ce paramètre est utilisé pour passer des arguments en ligne de commande au lancement de l'application. Dans la plupart des programmes, le nom utilisé pour `<identificateur>` est `argument` ou `args`, pour indiquer que la variable contient des arguments pour l'application.

2.2 Arguments en ligne de commande

2.2.1 Principes et utilisation

Une application Java étant un programme autonome, il peut être intéressant de lui fournir des paramètres ou des options qui vont déterminer le comportement ou la configuration du programme au lancement de celui-ci.

Rq: Les arguments en ligne de commande sont stockés dans un tableau de chaînes de caractères. Si vous voulez utiliser ces arguments sous un autre format, vous devez faire une conversion de type, du type String vers le type désiré lors de l'utilisation de l'argument.

Dans quels cas faut-il utiliser les arguments en ligne de commande ?

Les arguments en ligne de commande sont à utiliser au lancement d'une application dès qu'une ou des données utilisées à l'initialisation de votre programme peuvent prendre des valeurs variables en fonction de l'environnement. Par exemple :

- nom du port de communication utilisé dans le cas d'une communication avec un périphérique matériel.
- adresse IP d'une machine sur le réseau dans le cas d'une application client/serveur.
- nom d'utilisateur et mot de passe dans le cas d'une connexion à une base de données avec gestion des permissions d'accès.

Par exemple, dans le cas d'une application qui accède à une base de données, il faut en général fournir un nom d'utilisateur et un mot de passe pour ouvrir une session d'accès à la base. Des utilisateurs différents peuvent accéder à la base de données, mais avec des permissions différentes. Il peut donc exister plusieurs sessions différentes. Il n'est pas envisageable de créer une version de l'application pour chaque utilisateur.

De plus, ces informations sont susceptibles d'être modifiées. Il n'est donc pas judicieux de les intégrer dans votre code, car tout changement vous obligerait à modifier votre code source et à le recompiler et à détenir une version pour chaque utilisateur.

La solution à ce problème réside dans les arguments en ligne de commande.

Il suffit dans votre code d'utiliser le tableau d'arguments de la méthode main qui contient les variables (nom et mot de passe) de votre application.

Ensuite, selon l'utilisateur du programme, il faut au lancement de l'application par l'instruction `java`, faire suivre le nom de la classe principale par la valeur des arguments de ligne de commande de l'application.

2.2.2 Travail à réaliser

Le passage d'arguments à une application Java se fait au lancement de l'application par l'intermédiaire de la ligne de commande. L'exemple de programme suivant montre comment utiliser le passage d'arguments en ligne de commande dans une application Java.

```
/* Déclaration de la classe principale de l'application */
public class MaClasse
{

/* Déclaration de la méthode point d'entrée de l'application*/
public static void main(String [] args)
{

/* Affichage des arguments de la ligne de commande */
for (int i = 0 ; i < args.length; i++)
    System.out.println("Argument " + i + " = " + args[i]) ;

/* Conversion de deux arguments de la ligne de commande de
String vers int, puis addition des valeurs entières, et
affichage du résultat */
int somme;
somme = (Integer.parseInt(args[3])) + (Integer.parseInt(args[4]));
System.out.println("Argument 3 + Argument 4 = " + somme);
}}
```

Après compilation, le programme s'exécute avec la ligne de commande suivante :

```
java MaClasse arg1 bonjour "Hello World" 2 5
```

L'exécution du programme affiche les informations suivantes :

```
Argument    0    =    arg1

Argument    1    =    bonjour

Argument    2    =    Hello Wolrd

Argument    3    =    2

Argument    4    =    5

Argument    3    +    Argument 4 = 7
```