Introduction au « Java Development Kit »
Et quelques instructions en Java

Le *Java Development Kit* offre un ensemble d'outils de développement d'applications Java. Pour utiliser ces outils, JDK ne propose pas d'interface utilisateur, on doit donc écrire des lignes de commandes dans une fenêtre « terminal ». Des environnements plus sophistiqués existent, bien sûr, que vous étudierez ultérieurement, mais cette manière plus primitive de travailler vous permettra de bien comprendre les mécanismes mis en jeu lors de l'élaboration et de l'exécution des programmes.

1 Création des répertoires d'accueil des programmes Java

Soyez très soigneux dans toute la suite en ce qui concerne les noms de répertoires et de fichiers, principalement pour les majuscules et minuscules.
Vous devez créer les répertoires suivants pour accueillir le travail que vous ferez pendant les TPs de Java (aux feuilles de l'arbre, apparaissent certains des premiers fichiers que nous placerons dans les répertoires).

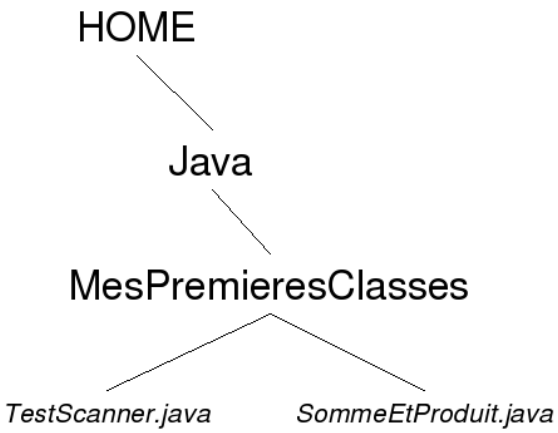

FIGURE 1 – Organisation des répertoires

Vous trouverez dans le dossier TP1 à l'adresse :

`https://moodle.umontpellier.fr/course/view.php?id=1584`

deux fichiers que vous copierez dans le répertoire `MesPremieresClasses` :
- `SommeEtProduit.java`,
- et `TestScanner.java`.

2 Modification des codes sources

Nous utiliserons xemacs ou kate pour l'édition des sources. Le lancement de xemacs se fait par la commande `xemacs &` ou en utilisant les menus ou icônes de votre environnement.
Ouvrez le code source du programme `SommeEtProduit.java` que nous vous rappelons au listing 1. Vous pouvez remarquer que le nom de la classe emballant ce programme est `SommeEtProduit`, c'est-à-dire le nom du fichier sans l'extension `.java`, ce n'est pas une coïncidence, c'est obligatoire.

Listing 1 – SommeEtProduit.java

```
1  package MesPremieresClasses;
2
3  import java.util.Scanner;
4
5  public class SommeEtProduit {
6    public static void main(String[] a) throws java.io.IOException
7    {
8        Scanner sc = new Scanner(System.in);
9
10       double x, y;
11       System.out.println("entrez deux doubles ");
12       x = sc.nextDouble();
13       y = sc.nextDouble();
14       double somme = x+y;
15       double produit = x*y;
16       System.out.println("somme = "+somme+" produit = "+produit);
17    }
18
19 }
```

3 CLASSPATH

Vous devez également mettre à jour une variable d'environnement nommée CLASSPATH. Dans votre fichier nommé `.bashrc`, placez la ligne suivante (n'oubliez pas de la faire suivre d'un « saut de ligne » si c'est la dernière ligne de votre fichier, sinon elle ne sera pas prise en compte) :
`export CLASSPATH=${HOME}/Java:.`

Pour prendre en compte cette modification, tapez la commande :
`. ~/.bashrc`
ou encore la commande
`source ~/.bashrc`

Java permet de désigner les classes de manière unique. Pour cela, Java concatène deux chemins : celui que contient `CLASSPATH` et celui qui correspond au nom du paquetage, par exemple, avec le programme qui précède : `${HOME}/Java` est concaténé avec `MesPremieresClasses`, ce qui donne le chemin absolu pour accéder à `SommeEtProduit`

$$\underbrace{\$HOME/Java/}_{\text{CLASSPATH}} \quad \underbrace{MesPremieresClasses/}_{\text{Paquetage}} \quad \text{SommeEtProduit}$$

La résolution d'une directive d'import fonctionne sur le même modèle.

4 Compilation

Le compilateur, qui va analyser le programme source et produire le programme en bytecode s'appelle `javac`.
Placez-vous dans une fenêtre terminal, et dans le répertoire `MesPremieresClasses`, la commande de compilation sera la suivante :
`javac SommeEtProduit.java`

Vous pouvez constater qu'un nouveau fichier `SommeEtProduit.class` est apparu dans votre répertoire `MesPremieresClasses`.

5 Exécution de programmes et entrée des données

5.1 Exécution

Notez que l'on ne peut exécuter (interpréter) que les classes qui contiennent une méthode `main`. La méthode `main` est la méthode principale de votre programme. Votre application commence par le début de cette méthode. Elle se terminera quand l'exécution sortira du bloc `main`.
Pour la classe `SommeEtProduit` par exemple, cela se fait par la commande :
`java MesPremieresClasses.SommeEtProduit`
à exécuter depuis le répertoire Java, qui contient le répertoire `MesPremieresClasses`.

5.2 Entrée des données

Lors de cette exécution, vous devez saisir deux nombres réels au clavier. Cela peut se révéler fastidieux lorsque le programme attend beaucoup de données.
Quatre autres solutions s'offrent à vous pour entrer des données.
Les deux premières consistent à créer un fichier contenant les données, sous la même forme que vous les auriez saisies au clavier. Par exemple, vous créez le fichier appelé `entree` qui contient (il se termine par une ligne vide) :

```
2
7,3
```

Puis vous pouvez appeler votre précédent programme de cette manière qui consiste à rediriger l'entrée du programme vers le fichier `entree` :
`java MesPremieresClasses.SommeEtProduit < entree`

Une autre solution pour utiliser le fichier consiste à l'ouvrir et à le parcourir à l'intérieur du programme avec des fonctions Java particulières. Nous la verrons plus tard.

La troisième solution consiste à utiliser les paramètres que l'on passe à la fonction `main`. Dans ce cas, on écrit un programme un peu différent, qui vous est présenté ci-après. Vous pouvez y observer que le paramètre `String[] a`, qui est un tableau de chaînes de caractères est exploité pour y récupérer successivement les deux nombres réels. Comme il s'agit d'un tableau de chaînes, la fonction `valueOf` de la classe `Double` est utilisée pour la conversion de `String` vers `double`.

```
package MesPremieresClasses;
public class SommeEtProduitEntreeParamMain {
  public static void main(String[] a) throws java.io.IOException
  {
      double x = Double.valueOf(a[0]);
      double y = Double.valueOf(a[1]);
      double somme = x+y;
      double produit = x*y;
      System.out.println("somme = "+somme+" produit = "+produit);
   }
}
```

Le programme est alors appelé de la manière suivante en ligne de commande (les paramètres sont sur la ligne de commande à la fin) :
`java MesPremieresClasses.SommeEtProduitEntreeParamMain 2 7.3`

Une quatrième solution consiste à écrire les valeurs à tester directement dans le programme. C'est une solution qui peut être utile pour des petits tests.

```
package MesPremieresClasses;
public class SommeEtProduitEnDur {
  public static void main(String[] a) throws java.io.IOException
  {
      double x = 2;
      double y = 7.3;
      double somme = x+y;
      double produit = x*y;
      System.out.println("somme = "+somme+" produit = "+produit);
   }
}
```

Vous pourrez remarquer le changement d'écriture des réels : on écrit 7,3 pour l'interpréteur (qui est francophone) et 7.3 pour le compilateur (qui est anglophone) avec notre installation actuelle.

6 Scanner

Regardez d'un peu plus près le programme `TestScanner.java`. Il utilise une instance de la classe `Scanner`, qui se trouve dans le paquetage `java.util` de l'API [1] Java fournie avec le `JDK`. La bibliothèque de classes fournie par Java est très grande et variée : elle permet la lecture et l'écriture dans différents médias, la communication réseau, la réalisation d'interfaces graphiques, fournit beaucoup de structures de données classiques, etc. Il est à noter que le chemin vers l'emplacement de la bibliothèque n'est pas à ajouter dans le `CLASSPATH`, il est connu grâce à une autre variable d'environnement. Quand on souhaite utiliser une classe de la bibliothèque, il suffit donc de connaître dans quel paquetage elle se trouve, ce qui permet soit d'importer la classe (par exemple : `import java.util.Scanner;` puis : `Scanner sc= ...`), soit d'utiliser le nom absolu de la classe (`java.util.Scanner sc=...`).
La classe `Scanner` permet l'analyse de texte, notamment de texte provenant de l'entrée standard (texte tapé sur la console). Quand on crée une instance de `Scanner`, on passe en paramètre du constructeur le texte à analyser : on peut lui passer un nom de fichier par exemple, ou bien un flux de texte comme `System.in`, qui permet l'analyse de l'entrée standard. L'analyseur permet de lire les éléments d'un texte les uns après les autres, pour cela, il est nécessaire de connaître le caractère séparateur des éléments dans le texte. Par défaut, il s'agit de l'espace.
L'analyseur permet aussi de traduire les éléments (suites de caractères) lus vers le type que l'on lui fournit : on peut demander à lire le prochain entier, et on récupère alors un élément de type entier, et pas de type chaîne. Bien sûr, l'analyse échoue si l'on demande la lecture d'un entier et que l'on fournit des caractères non traduisibles en entiers.
La classe `Scanner` dispose notamment des méthodes :

- `next()` qui retourne la prochaine chaîne lue dans le texte
- `nextInt()` qui retourne le prochain entier lu dans le texte
- `nextFloat()` qui retourne le prochain flottant lu dans le texte. Le caractère séparant la partie entière de la partie décimale est la virgule.

Compilez le fichier `TestScanner.java`, exécutez-le, et modifiez-le si vous le souhaitez. Vous remarquerez que la méthode *main* contient à la suite de sa déclaration une instruction qui peut paraître ésotérique : throws `IOException`. Ceci est dû au fait que les méthodes de la classe `Scanner` qui sont utilisées dans le main peuvent déclencher des erreurs (par exemple si on demande à lire un flottant et que la syntaxe des flottants n'est pas respectée par l'utilisateur (3.1 au lieu de $3,1$). Ces erreurs sont déclenchées sous la forme d'exceptions, vous verrez l'année prochaine en quoi cela consiste. En Java, quand on appelle une méthode `m` qui déclenche des exceptions, on doit :

- soit les propager (c'est-à-dire ne rien en faire et les laisser remonter soit vers l'utilisateur, soit vers une méthode appelante). Dans ce cas, la méthode `m` devient à son tour émittrice d'exceptions, et doit le signaler

1. Application Programming Interface

par l'instruction throws suivie du nom de l'exception (ou des exceptions). C'est ce qui est fait dans notre exemple : on propage l'exception de type `IOException`.

— soit d'attraper l'exception, ce qui arrête sa propagation.

7 Paquetages et visibilité

Ecrivez en Java le code correspondant au diagramme donné à la figure 2.

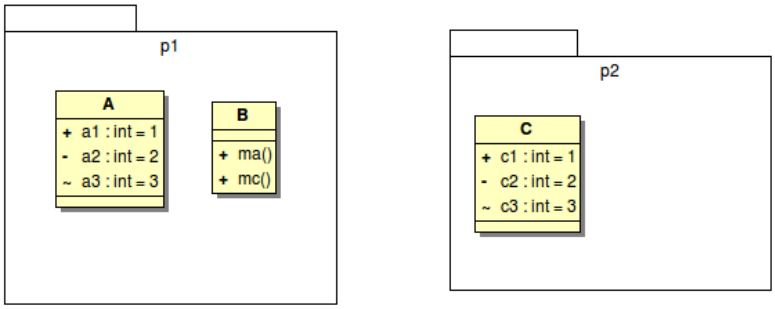

FIGURE 2 – Paquetages et visibilité

Compilez la classe A.
Dans le corps de la méthode `ma`, écrivez les lignes suivantes :

```
A a=new A();
System.out.println(a.a1);
System.out.println(a.a2);
System.out.println(a.a3);
```

Compilez la classe B. Quelle erreur est produite ? Pourquoi ? Commentez la ligne qui bloque la compilation. Recompilez. Prévoyez maintenant un paquetage appelé test et contenant une classe M avec une méthode main, qui sera le point d'entrée de notre programme. Pour exécuter, écrivez dans M.java :

```
package test;

import p1.B;

public class M {
/**
 * @param args pas d'arguments
 */
 public static void main(String[] args) {
  B b=new B();
  b.ma();
 }
}
```

Exécutez ce programme.

Dans le corps de la méthode `mc`, écrivez la ligne suivante :

```
C c=new C();
```

Compilez la classe B. Quelle erreur est produite ? Pourquoi ? Proposez une solution résolvant le problème. Recompilez.
Dans le corps de la méthode `mc`, ajoutez les lignes suivantes :

```
System.out.println(c.c1);
System.out.println(c.c2);
System.out.println(c.c3);
```

Compilez la classe B. Quelles erreurs sont produites ? Pourquoi ? Commentez les lignes qui bloquent la compilation. Recompilez.
Exécutez en ajoutant à la méthode main la ligne b.mc() ;

8 Application

Ecrivez en Java un programme qui demande la saisie sur l'entrée standard de 2 entiers e1 et e2, et 2 flottants f1 et f2, puis qui affiche le résultat de e1/e2 et f1/f2. Exécutez votre programme avec notamment :

- e1=6 et e2=4
- e1=6 et e2=0
- f1=6f et f2=4f

Qu'en déduisez-vous sur l'opérateur / ? Arrivez-vous à comprendre ce qui s'affiche lors d'une division par 0 ?