Prise en main de l'environnement Java

Le *Java Development Kit* offre un ensemble d'outils de développement d'applications Java. Pour utiliser ces outils, le JDK ne propose pas d'interface utilisateur, on doit donc écrire des lignes de commandes dans une fenêtre « terminal ».
Des environnements plus sophistiqués existent, nous vous proposons d'utiliser Eclipse. Dans la mesure où une grande partie des TP aura lieu à distance, vous pouvez également utiliser repl.it, ce qui permettra d'inviter votre enseignant à venir voir votre code. Si vous utilisez Eclipse, vous partagerez votre écran avec votre enseignant.
Les deux cas sont décrits ci-après pour un premier exemple.

1 Mise en place de votre environnement de travail

1.1 Cas 1 : vous voulez utiliser Eclipse

- (Si Eclipse n'est pas déjà installé) Installez Eclipse. Pour cela téléchargez l'installeur d'Eclipse et suivez les consignes présentes sur cette page : `https://www.eclipse.org/downloads/packages/installer` en choisissant d'installer "Eclipse IDE for Java Developpers".
- Lancez Eclipse.
- Choisissez votre "workspace" : cet espace de travail est un répertoire où vous rangerez vos projets Eclipse.
- Dans votre workspace, créez un nouveau projet Java (File → new → Java Project) que vous appellerez par exemple TP1-MPO1. Tout le contenu de votre projet sera alors dans un répertoire TP1-MPO1 placé dans votre répertoire de workspace.
- Vous trouverez à cette adresse `https://moodle.umontpellier.fr/course/view.php?id=1584` deux fichiers :
  - `SommeEtProduit.java`,
  - et `TestScanner.java`.

  Créez dans le répertoire src qui se trouve dans le répertoire qui correspond à votre projet Eclipse (par exemple dans TP1-MPO1/src) un nouveau répertoire appelé `tp1` (orthographié exactement ainsi) et placez-y ces 2 fichiers. Puis retournez dans Eclipse et rafraîchissez votre projet (clic droit sur le projet, puis refresh). Notez qu'il existe d'autres façons d'importer des éléments dans Eclipse.
- Normalement, il apparaît alors dans votre projet (partie gauche d'Eclipse en général) le package tp1, avec à l'intérieur les 2 classes SommeEtProduit et TestScanner. Aucune erreur ne devrait apparaître.

1.2 Cas 2 : vous utilisez repl.it

- (si vous n'avez pas déjà de compte repl.it) Inscrivez-vous sur `repl.it`.
- Créez un repl Java et pensez à y inviter votre enseignant.
- Ajoutez-y un répertoire nommé tp1
- Vous trouverez à cette adresse `https://moodle.umontpellier.fr/course/view.php?id=1584` deux fichiers :
  - `SommeEtProduit.java`,
  - et `TestScanner.java`.

  Téléchargez-les et déposez-les dans le répertoire tp1 du repl.

2 Compilation

Le compilateur, qui va analyser le programme source et produire le programme en bytecode, s'appelle `javac`. Nous allons dans un premier temps compiler `SommeEtProduit.java` qui contient la classe SommeEtProduit placée dans un package tp1. Son code est rappelé ici.

Listing 1 – SommeEtProduit.java

```
1  package tp1;
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 }
```

Vous pouvez remarquer que le nom de la classe est `SommeEtProduit`, c'est-à-dire le nom du fichier sans l'extension `.java`, ce n'est pas une coïncidence, c'est obligatoire. De même, il y a correspondance exacte entre le nom du package dans lequel est placée une classe et le nom du répertoire dans lequel est placé le fichier de cette classe.

2.1 Cas 1 : sous Eclipse

Eclipse compile en permanence, et souligne en rouge au fur et à mesure les éventuelles erreurs de compilation rencontrées (et en jaune les avertissements).
Eclipse n'affiche pas par défaut dans son interface la présence des fichiers compilés. Pour les voir, placez-vous (à l'extérieur d'Eclipse par exemple via un terminal) dans le répertoire qui correspond à votre projet Eclipse (par exemple dans TP1-MPO1). Vous verrez qu'à côté du répertoire src dans lequel se trouvent vos sources, il y a un répertoire bin dans lequel se trouvent vos classes compilées (les .class).

2.2 Cas 2 : sous repl.it

Le code est également régulièrement compilé, mais il n'est pas toujours clair de savoir si la compilation a eu lieu ou pas. Le plus simple est de la faire à la main, dans la fenêtre placée à droite, qui vous présente une console, dans laquelle vous êtes dans un répertoire contenant le répertoire tp1. Pour compiler, lancer la commande : `javac tp1/SommeEtProduit.java`. Avec la commande `ls tp1`, vérifiez que le fichier compilé suffixé par .class est bien apparu.

3 Exécution de programmes et entrée des données

3.1 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`.

3.2 Cas 1 : sous Eclipse

Appuyer sur la flèche verte pour lancer l'exécution.

3.3 Cas 2 : sous repl.it

Si le repl est bien configuré, on peut appuyer sur la flèche verte. Sinon, on exécute avec la commande : `java tp1.SommeEtProduit`. Pour configurer l'exécution avec la flèche verte, créez dans votre repl un fichier nommé .repl, et ajoutez-y :

```
1 run = "javac␣tp1/*.java;java␣tp1.SommeEtProduit;"
```

Cela permet de compiler puis d'exécuter SommeEtProduit.

3.4 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 tp1.SommeEtProduit < entree`

Sous Repl.it : utilisez directement dans la console la commande ci-dessus.

Sous Eclipse : il faut aller dans les configurations d'exécution. Dans le menu run, choisissez Run configurations. Vous devriez alors trouver une configuration SommeEtProduit. C'est elle qui indique comment il faut exécuter SommeEtProduit. Sélectionnez cette configuration. Commencez par dupliquer cette configuration (pour ne pas la perdre) avec le bouton en haut à gauche, à côté de la croix rouge. Renommez la configuration dupliquée, par exemple le nouveau nom peut être : "SommeEtProduit avec fichier d'entrée", puis sélectionnez l'onglet common. Vous pouvez choisir ici un fichier d'entrée (et aussi de sortie, ce que l'on ne fera pas ici). Cochez l'utilisation d'un fichier d'entrée (input file) puis choisissez le fichier d'entrée. Sauvegardez. Maintenant, quand vous voudrez exécuter SommeEtProduit, vous pourrez choisir l'une ou l'autre des 2 configurations en allant dans Run, puis Run configurations.

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 verrons cette solution 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`.

```
1  package tp1;
2  public class SommeEtProduitEntreeParamMain {
3    public static void main(String[] a) throws java.io.IOException
4    {
5        double x = Double.valueOf(a[0]);
6        double y = Double.valueOf(a[1]);
7        double somme = x+y;
8        double produit = x*y;
9        System.out.println("somme␣=␣"+somme+"␣produit␣=␣"+produit);
10     }
11 }
```

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 tp1.SommeEtProduitEntreeParamMain 2 7.3`

Sous Repl.it : utilisez directement dans la console la commande ci-dessus.

Sous Eclipse : il faut de nouveau aller dans les configurations d'exécution. Dans le menu run, choisissez Run configurations. Créez une nouvelle configuration (bouton en haut complètement à gauche). Puis allez dans l'onglet Arguments. Vous y trouverez une fenêtre pour taper les Program Arguments. Saisissez dans cette zone : `2 7.3`. Attention, ne confondez pas avec la zone VM arguments (qui est pour les arguments de la machine virtuelle). Puis exécutez.
Une quatrième solution consiste à écrire les valeurs à tester directement dans le programme. C'est une solution qui est à privilégier pour faire vos tests et en garder une mémoire, tant que vous n'aurez pas vu de meilleurs manières de tester vos programmes (le test unitaire avec des méthodes plus adéquates sera abordé dans une autre UE).

```
package tp1;
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);
   }
}
```

Selon votre installation, 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 certaines installations.

4 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.
Quand on souhaite utiliser une classe de la bibliothèque, il suffit 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.

1. Application Programming Interface

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 émettrice d'exceptions, et doit le signaler 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.

5 Paquetages et visibilité

Ecrivez en Java le code correspondant au diagramme donné à la figure 1 en laissant vide le corps des méthodes ma() et mc() de la classe B.

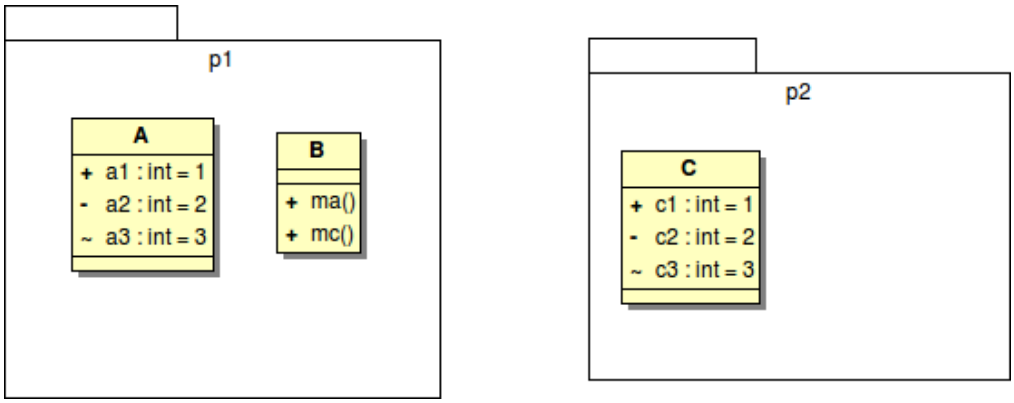

FIGURE 1 – Paquetages et visibilité

Compilez la classe A.
Dans la classe B, définissez ainsi la méthode `ma` :

```
1 public void ma(){
2   A a=new A();
3   System.out.println(a.a1);
4   System.out.println(a.a2);
5   System.out.println(a.a3);
6 }
```

Compilez la classe B. Quelle erreur est produite ? Pourquoi ? Commentez la ligne qui bloque la compilation. Recompilez.
Prévoyez maintenant un paquetage appelé test (non représenté sur la figure) et contenant une classe M avec une méthode main, qui sera le point d'entrée de notre programme. Le fichier M.java contiendra :

```
1 package test;
2 import p1.B;
3
4 public class M {
5     /**
6      * @param args pas d'arguments
7      */
8      public static void main(String[] args) {
9           B b=new B();
```

```
10              b.ma();
11          }
12  }
```

Compilez et exécutez la classe M.
La méthode `mc` de la classe B est maintenant définie ainsi :

```
1  public void mc(){
2    C c=new C();
3  }
```

Compilez de nouveau la classe B. Quelle erreur est produite ? Pourquoi ? Proposez une solution résolvant le problème. Recompilez.
Modifiez la méthode `mc` ainsi :

```
1  public void mc(){
2    System.out.println(c.c1);
3    System.out.println(c.c2);
4    System.out.println(c.c3);
5
6  }
```

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

6 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 ?