Chapitre 3

Le programme en JAVA

L'objectif de ce chapitre est d'écrire ses premiers programmes JAVA.
Mais aussi d'introduire et d'anticiper sur la structuration d'un programme JAVA.

Nous aborderons donc des notions qui seront détaillées par la suite.

Votre objectif sera d'installer sur votre PC un environnement de programmation Java et de vous exercer car la programmation est une discipline pratique.

1. Structure générale d'un programme Java

Un programme Java est un fichier source .java qui contient une classe publique Java dans laquelle est définie la méthode main :

```
public class Exemple
{
   // Attributs de la classe Exemple

   public static void main(String a_args[])
 // a_args sont les parametres de lancement du programme Java
   {
      // Déclaration des variables propre au main

      // Instructions du programme principal
            // Création d'un objet
            // Appel de méthode de l'objet créé

            // instructions sans l'utilisation d'objet

   }

   // Méthodes de la classe Exemple

   public void m1 ()
      {
        // Déclaration des variables propre à la méthode m1

        // Instructions
      }


}// Exemple


class C1   // Classe privée de définition d'un objet
{
     // Attributs de la classe C1

     // Méthodes de la classe C1
}

class C2 // Classe privée de définition d'un objet
{
     // Attributs de la classe C2

     // Méthodes de la classe C2

}
```

Le point d'entrée d'un programme Java est la méthode **main** de la classe principale (Exemple).

La méthode main prend en entrée les paramètres de la ligne de commande d'exécution du programme (une fois le programme compilé) :
$ java Exemple p1 p2 p3

Déclaration des variables propre au main

Cette zone contient les noms des variables qui sont utilisées dans la méthode main.

Instructions du programme principal

Cette zone représente le code proprement dit du programme. Cette zone utilise les variables définies au-dessus ainsi que le tableau de paramètres du programme (a_args). Généralement elle crée un ou plusieurs objets et appelle les méthodes de ces objets créés.

La définition de la méthode main est :

```
public static void main(String a_args[])
{

}
```

Cette méthode est "**public**" afin que la JVM puisse accéder à la méthode main.

Cette méthode est "**static**" afin que la JVM puisse appeler la méthode main sans passer par la création d'un objet.

Cette méthode est "**void**" car elle ne retourne pas de valeur.

Cette méthode s'appelle "**main**" car c'est le moyen par défaut qu'a la JVM pour connaître quel est le nom de la méthode à utiliser pour lancer le programme.

Cette méthode a en paramètre **String[]** qui correspond à un tableau de chaîne de caractère contenant les paramètres de la ligne de commande d'exécution.

```
public class Exemple
{
   public static void main(String a_args[])
   {
      System.out.println("Paramètre 1 : " + a_args[0]);
      System.out.println("Paramètre 2 : " + a_args[1]);
   }
}



Pour compiler :
$ javac Exemple.java



$ java Exemple TOTO 123
Paramètre 1 : TOTO
Paramètre 2 : 123
```

Nous verrons plus tard l'usage des tableaux en Java.

Le programme a pour nom le nom de la classe : class Exemple

L'instruction *System.out.println* permet d'écrire une chaîne de caractère à l'écran. Nous verrons plus tard la signification exacte de cette syntaxe bien particulière.

L'opérateur "+" réalise la concaténation de deux chaînes de caractère.

Même si le programme n'a pas de paramètre, il est indispensable de déclarer le tableau de String dans la méthode main.

Les commentaires sont des lignes de texte qui seront ignorés par le compilateur.
En Java, est en commentaire tout texte encadré par "//" et la fin de ligne, ou tout texte de plusieurs lignes encadrés par /* et par */.

Exemple :

```
1.public class Exemple
2.{
3.   public static void main(String a_args[])
4.   {
5.      // Zone de déclaration des variables
6.      int x, y;
7.      int z;
8.
9.      // Zone des instructions
10.     x = Integer.parseInt(a_args[0]);
11.     y = Integer.parseInt(a_args[1]);;
12.     z = x + y;
13.     System.out.println("Le résultat est : " + z);
14.
15.     Additionneur add = new Additionneur(x,y);
16.     add.calculer();
17.     add.afficher();
18.
19.   }
20.  }
21.
22.  class Additionneur
23.  {
24.     int vx;
25.     int vy;
26.     int vz;
27.
28.     public Additionneur(int px, int py)
29.     {
30.       vx = px;
31.       vy = py;
32.     }
33.
34.     public void calculer()
35.     {
36.       vz = vx + vy;
37.     }
38.
39.     public afficher()
40.     {
41.       System.out.println("Le résultat est : " + vz);
42.     }
43.
44.
```

ligne 6 : déclaration de deux variables x et y de type entières.
ligne 7 : déclaration d'une variable z de type entière

ligne 10 : instruction d'affectation : x prend la valeur entière du 1er paramètre
ligne 11 : instruction d'affectation : y prend la valeur entière du 2ème paramètre

ligne 12 : instruction d'affectation et expression arithmétique qui réalise l'addition de x et y, et affecte le résultat à la variable z.

ligne 13 : instruction qui affiche à l'écran le résultat.

La même chose mais en utilisant un objet :

ligne 15 : création d'un objet Additionneur
ligne 16 : appel de la méthode de calcul
ligne 17 : affichage du résultat du calcul

ligne 22 à 42, définition de la classe d'un Additionneur
- ligne 24 à 26 : les attributs de la classe
- ligne 28 : le constructeur qui permet de mémoriser dans l'objet les valeurs à calculer
- ligne 34 : méthode de calcul qui réalise le calcul et mémorise le résultat dans l'objet
- ligne 39 méthode qui affiche le résultat du calcul

On compile le programme :

```
$javac Exemple.java
```

On exécute le programme :

```
$java Exemple 12 23
Le résultat est : 35
```

2. La compilation du programme Java

Soit le programme suivant écrit dans le fichier :

Bonjour.java

```
public class Bonjour
{
   public static void main(String a_args[])
   {
      Terminal.ecrireStringln("Bonjour tout le monde!");
   }
}
```

Voir sur le site http://jacques.laforgue.free.fr l'exemple **Exemple00_Bonjour**

Commande de compilation :

```
D:\CNAM\Exemples\Exemple1>javac Bonjour.java
```

Commande d'exécution

```
D:\CNAM\Exemples\Exemple1>java Bonjour
Bonjour tout le monde!
```

Exemple d'une erreur de compilation :

```
public class Bonjour
{
   public static void main(String a_args[])
   {
      Terminal.ecrireStringln("Bonjour tout le monde! "+l_monNom);
   }
}
```

```
d:\Jacques\CNAM\SITE\tmp>javac Bonjour.java
Bonjour.java:5: cannot find symbol
symbol  : variable l_monNom
location: class Bonjour
     Terminal.ecrireStringln("Bonjour tout le monde! "+l_monNom);
                                                       ^
1 error
```

On modifie le programme :

```
public class Bonjour
{
   public static void main(String a_args[])
   {
      String l_monNom = a_args[0];
      Terminal.ecrireStringln("Bonjour tout le monde! "+l_monNom);
   }
}
```

Pas d'erreur de compilation

Exemple d'une erreur d'exécution :

```
d:\Jacques\CNAM\SITE\tmp>java Bonjour
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
 at Bonjour.main(Bonjour.java:5)
```

Nous avons oublié de passer le paramètre.

En passant le paramètre :

```
d:\Jacques\CNAM\SITE\tmp>java Bonjour Dupont
Bonjour tout le monde! Dupont
```

En règle générale, on préfère mettre les fichiers .class dansun repertoire séparé : bin

Voir sur le site http://jacques.laforgue.free.fr l'exemple **Exemple00_BonjourBis**

Commande de compilation :

```
D:\CNAM\Exemples\Exemple1>javac -d bin Bonjour.java
```

Commande d'exécution, dans le répertoire bin :

```
D:\CNAM\Exemples\Exemple1\bin>java Bonjour
Bonjour tout le monde!
```

3. Les opérations de lecture et d'écriture

Pour réaliser nos premiers programmes informatiques, nous avons besoin d'écrire des valeurs à l'écran mais aussi saisir des valeurs de type entier, double, caractère, booléenne ou chaîne.

Pour cela, j'ai créé des méthodes qui cachent l'implémentation de ces opérations, dont le détail sera vu plus tard.

Ces méthodes sont définies dans une classe Java Terminal qui est définie dans le fichier Terminal.java. Ce fichier doit être à côté de votre programme.

Cela me permet ainsi d'introduire la création et l'appel des méthodes en Java, dont un chapitre est entièrement consacré.

La classe **Terminal.java :**

Voir sur le site http://jacques.laforgue.free.fr l'exemple
Outil00a_TerminalSansPackage

L'objectif de la présentation de cette classe Terminal n'est pas de comprendre l'ensemble du code qui s'y trouve mais d'utiliser les méthodes de lecture et d'écriture des différents types :

```
public class Exemple
{
   public static void main(String a_args[])
   {
      String  val_string;
      int     val_int;
      double  val_double;
      char    val_char;
      boolean val_bool;

      Terminal.ecrireString("Saisir une chaine : ");
      val_string = Terminal.lireString();
      Terminal.ecrireStringln(val_string);

      Terminal.ecrireString("Saisir un entier : ");
      val_int    = Terminal.lireInt();
      Terminal.ecrireIntln(val_int);

      Terminal.ecrireString("Saisir un double : ");
      val_double = Terminal.lireDouble();
      Terminal.ecrireDoubleln(val_double);

      Terminal.ecrireString("Saisir un booléen : ");
      val_bool = Terminal.lireBoolean();
      Terminal.ecrireBooleanln(val_bool);

      Terminal.ecrireString("Saisir un caractère : ");
      val_char = Terminal.lireChar();
```

```
        Terminal.ecrireCharln(val_char);
    }
}
```

Commentaires :
- Il ne peut y avoir d'erreur de saisie pour une String
- Pour un entier si vous ne saisissez pas un entier, il y a une erreur (exception)
- Pour un double, si vous ne saisissez pas un entier ou un double, il y a une erreur
- Pour un booléen, toute valeur autre que "True" ou "true" est false.
- Pour un caractère seul le 1er caractère est pris en compte

La class Terminal est constitué de méthodes. Cette classe joue ici le rôle d'une bibliothèque de méthode.

On appelle les méthodes de la bibliothèque suivant la syntaxe suivante :

```
<classe>.<méthode>
```

4. Un premier exemple complet

Voir sur le site http://jacques.laforgue.free.fr l'exemple **Exemple01a_ConversionEuro**

On se propose d'écrire le programme qui consiste à demander une valeur en euro, de convertir cette valeur en franc et d'afficher à l'écran la valeur en franc.

L'algorithme est le suivant :

```
Debut
   Ecrire le nom du programme;
   Saisir la valeur en euro;
   Convertir la valeur saisie en francs sur la base de 6.559;
   Afficher la valeur en francs;
Fin
```

Le code est le suivant :

```
public class Exemple1
{
    public static void main(String a_args[])
    {
        double l_euros;
        double l_francs;

        Terminal.ecrireStringln("PROGRAMME DE CONVERSION EN EURO");
        Terminal.ecrireStringln("");
        Terminal.ecrireString("Veuillez saisir la somme en euros: ");
        l_euros = Terminal.lireDouble();

        l_francs = l_euros * 6.559;
        Terminal.ecrireStringln("La somme convertie en francs est :
"+l_francs);
    }
```

```
}
```

Exécution du programme :

```
java Exemple1
PROGRAMME DE CONVERSION EN EURO

Veuillez saisir la somme en euros: 10
La somme convertie en francs est : 65.59
```

```
java Exemple1
PROGRAMME DE CONVERSION EN EURO

Veuillez saisir la somme en euros: 2,5
Exception in thread "main" TerminalException
 at Terminal.exceptionHandler(Terminal.java:117)
 at Terminal.lireDouble(Terminal.java:49)
 at Exemple1.main(Exemple1.java:12)
```

Dans ce cas on a saisie une valeur 2,5 qui ne peut pas être convertie en double.

On décortique le programme:

```
double l_euros;
double l_francs;
```

Ces deux lignes réalisent la déclaration des variables locales du programme.

```
Terminal.ecrireStringln
```

Cette instruction permet d'afficher une chaîne de caractère dans la fenêtre d'exécution. Elle n'est pas une instruction prédéfinie de Java mais une instruction crée par votre serviteur pour faciliter l'écriture de nos programmes. Nous verrons plus tard le contenu de ce composant.

```
l_euros = Terminal.lireDouble()
```

Cette instruction attend la saisie au clavier d'un texte qui doit être un double bien formé, suivi d'un retour chariot. La valeur saisie est affectée à la variable l_euros.

```
"La somme convertie en francs est : "+l_francs
```

Cette instruction réalise la concaténation entre une chaine de caractère et une valeur en double.

Il est à noter que ce programme n'est pas robuste dans la mesure où, comme on l'a vu précédemment, si l'opérateur saisie une valeur qui n'est pas un double, le programme "plante". Nous verrons plus tard que nous pourrons, grâce aux exceptions, résoudre ce problème en demandant à l'opérateur de saisir une valeur tant que la valeur saisie n'est pas un double.

5. Le même exemple avec l'utilisation d'un "Formulaire"

Voir sur le site http://jacques.laforgue.free.fr l'exemple **Exemple01b_ConversionEuro**

Pour faciliter la réalisation d'un programme informatique qui demande à un utilisateur de saisir des informations dans une "belle" interface homme-machine, j'ai créé la classe **Formulaire** dont voici un exemple de l'utilisation.

Nous voulons faire le même programme que précédemment mais avec une Ihm.

Pour utiliser les formulaires il faut copier/coller le répertoire "fr" de Outil02_FormulaireIHM dans votre répertoire et mettre l'instruction d'import dans vos fichiers

```
import fr.cnam.ihm.*;
```

Le code est le suivant :

```
  //  Programme de conversion de euros en francs (sans objet)<br>En
utilisant la classe Formulaire

  import fr.cnam.ihm.*;

  // La classe implemente l'interface FormulaireInt pour qu'elle
  //  puisse realise les actions declenchees dans un formulaire
  //
  public class Exemple01b implements FormulaireInt
  {
      public static void main(String a_args[])
      {
          // Creation de l'objet d'exemple
          Exemple01b ex = new Exemple01b();

          // Creation du formulaire en passant en parametre
          // l'objet d'exemple
          Formulaire form = new Formulaire("DEVISE",ex,500,100);

          // Creation des elements de l'ihm
          form.addLabel("Faire la conversion d'euros en francs :");
          form.addText("SOMME_EURO","Euros",true,"");
          form.addButton("CONVERTIR","Convertir");
          form.addText("SOMME_FRANCS","Francs",false,"");
          form.afficher();
      }

      // Methode appelee dans le formulaire quand on utilise un des
      //  boutons
      //    nomSubmit est le nom du bouton utilise
      public void submit(Formulaire form,String nomSubmit)
      {
          // On teste le nom du bouton (il pourrait y avoir
          //   plusieurs boutons
          if (nomSubmit.equals("CONVERTIR"))
              {
                  // Recuperation de la valeur saisie dans le champ
                  String s = form.getValeurChamp("SOMME_EURO");
                  double d = Double.parseDouble(s);
                  // Calcul
                  double l_euros = d* 6.559;
                  String f = l_euros+"";
                  // On met le resultat dans un champ
                  form.setValeurChamp("SOMME_FRANCS",f);
              }
```

```
    }
}
```

Commentaires :

1/ utilisation du package fr.cnam.ihm dans lequel sont définis la classe **Formulaire** et l'interface **FormulaireInt**.

```
import fr.cnam.ihm.*;
```

2/ Utilisation d'une interface et donc d'un objet passé en paramètre qui implémente la méthode **submit**. (Nous verrons en NFA032 la définition et l'utilisation d'une interface plus précisemment). Dans le cadre de NFA 031 nous n'utiliserons la notion d'interface que dans ce cas là).

```
Formulaire form = new Formulaire("DEVISE",ex,500,100);
```

3/ Les lignes suivantes concernent la création des éléments du formulaire (label, zone de saisi de texte, bouton pour réaliser une action).

4/ La méthode submit est appelé à chaque fois que l'on clique dans un bouton du formulaire. Cette méthode prend en entrée le formulaire créé et le nom du bouton utilisé.

```
public void submit(Formulaire form,String nomSubmit)
```

5/ on récupère la valeur saisie

```
String s = form.getValeurChamp("SOMME_EURO");
```

6/ On réalise le calcul (comme précédemment), puis on affiche le résultat dans un champ de l'IHM

```
form.setValeurChamp("SOMME_FRANCS",f);
```

6. Un deuxième exemple complet

6.1. Enoncé

Faire le programme java qui fait la résolution de l'équation du 2émé degré :

$y = ax^2 + bx + c = 0$

Les coefficients sont saisis par l'opérateur.
Le programme affiche le texte "Pas de solution" s'il n'y a pas de solution dans le monde réel, affiche les deux racines ou une seule.
Le programme vérifie si le résultat est juste.
Vous utilisez la classe Terminal pour les instructions d'entrée sortie.

Théorème

Soit (E) : $ax^2 + bx + c = 0$ une équation du second degré où a est un nombre réel non nul, b, c sont deux nombres réels, et Δ son discriminant.

- si $\Delta < 0$, alors l'équation (E) n'a pas de solution dans $\mathbb{R}$;
- si $\Delta = 0$, alors l'équation (E) a une unique solution x_0 dans $\mathbb{R}$, définie par : $x_0 = -\frac{b}{2a}$;
- si $\Delta > 0$, alors l'équation (E) a dans $\mathbb{R}$ deux solutions distinctes x_1 et x_2 définies par : $x_1 = \frac{-b-\sqrt{\Delta}}{2a}$ et $x_2 = \frac{-b+\sqrt{\Delta}}{2a}$.

Avec $\triangle = b^2-4ac$

La méthode qui calcule la racine carré d'un nombre est:

```
double Math.sqrt(double d);
```

Quelques exemples de résultats :
a= 1 b= -1 c= -1 x1 = 1.618 x2 = -0.618 (nombre d'or)

a= 1 b= 2 c= 1 x1=x2= -1
a= 1 b= 1 c= 1 pas de solution

6.2. Correction en mode Terminal

Voir sur le site http://jacques.laforgue.free.fr l'exemple **Exemple03a_Equ2Degre**

Commentaires :

Le traitement de résolution de l'équation du second degré est codé sous la forme d'une méthode **resoudre** appartenant à une classe : **Equ2Degre**.

Ce traitement correspond à l'algorithme que nous avons vu dans le 1er cours d'introduction.

Ce traitement prend en entrée les coefficients et les résultats du traitement sont des attributs de la classe :

String erreur : si différent de null alors correspond à un cas d'erreur.

double x1 et x2 : les solutions de l'équation (si pas d'erreur).

Le programme principal , méthode **main**, est une boucle sur l'utilisation du traitement.

Ce programme affiche à l'écran les résultats et réalise la vérification du calcul.

6.3. Correction avec l'utilisation d'un formulaire

Voir sur le site http://jacques.laforgue.free.fr l'exemple **Exemple03b_Equ2Degre**

Avec l'utilisation d'un formulaire avec une zone de texte qui sert d'historique des calculs :

Voir sur le site http://jacques.laforgue.free.fr l'exemple **Exemple03c_Equ2Degre**

7. Environnements de programmation

7.1. En mode commande

Le pré-requis est l'installation sur votre ordinateur d'un IDE (Integrated Development Environment) Java.
Vous pouvez télécharger la version de Java situé sur le site.
http://jacques.laforgue.free.fr/SITE_NFA031/Site/Logiciels.phtml

Temps de téléchargement de plusieurs minutes en fonciton de votre débit.

Il faut ajouter dans la variable d'environnement (Windows ou Linux) **PATH** le chemin d'accès au répertoire bin du JDK qui contient le compilateur (javac) et la JVM (java).

Vous créez un répertoire dans lequel vous voulez créer votre programme java.

Pour windows, dans ce répertoire vous créez un lien de l'utilitaire d'invite de commande (situé dans Tous les programmes > Accessoires).
Puis clic-droit > Propriétés : effacer le texte de "Démarrer dans".

Créer votre fichier texte, par exemple Prog.java

Puis dans l'invite de commande vous tapez : javac Prog.
Puis pour exécuter : java Prog.

Sous Linux, vous utilisez l'utilitaire d'exécution de commande.

Pour éditer le programme Java, vous utilisez un éditeur de texte simple (pas word) comme Bloc-Notes, WordPad ou emacs (disponible sur le site).

Pour ne pas être obligé de taper à chaque fois les commandes de compilation et d'exécution, vous pouvez créer des fichiers de commande : en windows des .bat, en Linux des .sh.

7.2. Avec Eclipse

L'utilisation de eclipse à un avantage et un inconvénient.

L'avantage est qu'il permet de créer ces lignes de code avec rapidité et efficacité grâce à la complétion automatique, à la génération de code et surtout grâce à la compilation à la volée qui permet de connaitre en temps réel les erreurs de codage.

Le désavantage est qu'il faut maitriser cet environnement. Ce qui dans un premier temps peut paraître lourd, surtout pour faire ses tous premiers programme Java. On lui préfère alors le mode commande.

Pour utiliser eclipse, il faut installer une version d'éclipse que l'on trouve en téléchargement sur le site :
https://www.eclipse.org/downloads/

Vous pouvez prendre la dernière version : Oxygen (46,4 Mo).
Lors de l'installation, choisir "Eclipse IDE for Java Developers".
Eclipse s'installe dans <user>/eclipse

Quand on lance Eclipse, il demande la localication du Workspace.

Le Workspace est un répertoire de travail d'éclipse dans lequel on peut ou non y mettre les sources.

Vous créez un nouveu projet (File>Java project) puis, soit vous créez un nouveau programme (1er cas), soit vous importer un programme existant (2ème cas) comme par exemple un des exemples du site, soit vous utilisez les sources là où ils sont.

1er cas :

Pour importer, clic droit sur le répertoire src, puis choisir Generel > File Systems
next, puis coller dans From directory le répertoire de l'exemple, suivi de RC,
puis sélectionner la case à cocher de l'exemple,
puis finish.

Pour exécuter, sélectionner le fichier .java contenant la classe contenant la méthode main, puis exécuter (bouton run).

2ème cas :

Pour travailler sur les sources, quand vous créez le projet (File>Java project)
Vous décochez "Use default location" puis avec Browse vous choisissez le répertoire des sources. Puis, Finish.
Dans ce cas, quand vous supprimer le projet d'Eclipse, vous devez le faire sans supprimer les sources.

Dans les 2 cas :

Pour savoir, si le projet est dans le 1er cas ou le 2ème cas, clic-droit sur projet>Propriétés : Location.

Le mode debug (à voir en séance).

Création d'une classe (à voir en séance).