1 Structure d'un programme Java

1.1 Méthode de développement d'un logiciel

Voici une démarche par étape qu'on pourra suivre pour développer un logiciel :

1. Bien comprendre le problème à résoudre et ce qu'on nous demande de réaliser.
2. Imaginer une solution qui résoudra le problème :
   (a) Découper les problèmes complexes en problèmes plus simples à résoudre.
   (b) Trouver des algorithmes pour résoudre les problèmes simples.
   (c) Écrire ces algorithmes en pseudocode.
3. Programmer la solution :
   (a) Écrire le code qui correspond au pseudocode.
   (b) Compiler le code.
   (c) Exécuter le programme et vérifier sommairement son fonctionnement.
4. Tester le programme :
   (a) Spécifier différents jeux d'essais pour tester les différents cas d'utilisation du programme.
   (b) Effectuer les essais.
   (c) Corriger les erreurs détectées.

1.2 Édition du code

On peut écrire le code dans un éditeur de texte simple comme Notepad. Par contre, on recommande l'utilisation d'un environnement de développement intégré comme BlueJ, Eclipse, ou n'importe quel autre environnement. Dans ce cours, seul l'environnement BlueJ sera supporté.

Les environnements de développement intégrés ont l'avantage de reconnaître la syntaxe Java en plus de permettre la compilation et l'exécution des programmes facilement.

Voici un exemple de programme Java :

```
/**
* Premier programme Java.
*
* @author VotrePrénom VotreNom
*/
public class PremierProgrammeJava {

    /**
     * Programme principal
     */
     public static void main (String[] args) {

         // Déclaration d'une variable
         String prenom;

         // Affichage d'un texte
         System.out.print("Veuillez saisir votre prenom: ");

         // Saisir une valeur à travers le clavier
         prenom = Clavier.lireString();

        // Affichage d'un texte
        System.out.print("Merci ");

        // Affichage de la valeur d'une variable
        System.out.println(prenom);

        // Affichage d'un texte
        System.out.println("Fin du programme.");

  }

}
```

1.3 Compilation

La phase de compilation consiste à convertir le code source sous une forme exécutable par l'ordinateur. Cette phase est réalisée par un compilateur intégré dans l'environnement de développement. Le compilateur valide la syntaxe du code source, interprète les instructions et les exprime dans une forme exécutable.

1.4 Exécution

La phase d'exécution est réalisée par la machine virtuelle Java. Cette phase consiste à exécuter les instructions fournies par le compilateur.

2 Les constantes et les types

2.1 Types

Un type peut être assimilé à un ensemble de valeurs. Dans ce cours, nous allons utiliser les types Java suivants :

- Les entiers :
  - `byte` (1 octet) : -128, . . . , 127
  - `short` (2 octets) : -32 768, . . . , 32 767
  - `int` (4 octets) : -2 147 483 648, . . . , 2 147 483 647
  - `long` (8 octets) : -9 223 372 036 854 775 808, . . . , 9 223 372 036 854 775 807
- Les réels :
  - `float` (4 octets).
  - `double` (8 octets).

  Le calcul de l'ensemble des valeurs pour les réels peut être complexe et ne fait pas partie des objectifs de ce cours. Il suffit de savoir que le type `double` permet de manipuler des valeurs avec une précision plus élevée qu'avec le type `float` et qu'il occupe un espace mémoire deux fois plus grand.
- Les caractères :
  - `char` (2 octets) : 'a', . . . , 'b', . . . , 'A', . . . , 'Z', '0', . . . , '9', '!', '$', '|', . . . , '\n', '\\', '\t', . . .
- Les chaînes de caractères (ce n'est pas un type de base, mais possède certaines caractéristiques des types) :
  - `String` : null, "", "x", "Bonjour", . . .
- Les booléens :
  - `boolean` : true, false

2.2 Constantes

Dans un programme Java on peut écrire directement des *valeurs* (sauf pour les types `byte` et `short`). Voici des exemples de valeurs :

- La valeur `23` est de type `int`
- La valeur `23L` est de type `long`
- La valeur `56.49` est de type `double`
- La valeur `56.49D` est de type `double`
- La valeur `56.49d` est de type `double`
- La valeur `56.49F` est de type `float`
- La valeur `56.49f` est de type `float`
- La valeur `'b'` est de type `char`
- La valeur `'\n'` est de type `char`
- La valeur `true` est de type `boolean`
- La valeur `"Paul"` est de type `String`
- La valeur `3` est de type `int`
- La valeur `'3'` est de type `char`
- La valeur `"3"` est de type `String`
- La valeur `"true"` est de type `String`

Toutes ces valeurs sont des *constantes*. Il y a des valeurs qui ne sont pas définies dans Java, par exemple :

- La valeur `2 147 483 648` n'est pas un `int` car la dernière valeur pour le type `int` est `2 147 483 647`.
- La valeur `9 223 372 036 854 775 808L` n'est pas un `long` car la dernière valeur pour le type `long` est

`9 223 372 036 854 775 807L`.

— La valeur `'ab'` n'est ni de type `char` ni de type `String`.

Voici un programme qui affiche des constantes :

```
public class TesterConstantes {

  public static void main ( String [] arsg ) {

    System.out.println( 23 );
    System.out.println( 23L );
    System.out.println( 56.49 );
    System.out.println( 56.49D );
    System.out.println( 56.49d );
    System.out.println( 56.49F );
    System.out.println( 56.49f );
    System.out.println( 'b' );
    System.out.println( '\n' );
    System.out.println( true );
    System.out.println( "true" );
    System.out.println( "Paul" );
    System.out.println( 3 );
    System.out.println( '3' );
    System.out.println( "3" );

  }

}
```

Voici une exécution de ce programme (remarquez la présence des deux lignes vides au milieu) :

```
23
23
56.49
56.49
56.49
56.49
56.49
b


true
true
Paul
3
3
3
```

2.3 Constantes symboliques

On peut associer un nom symbolique à une constante si on veut lui donner un sens significatif (c'est souvent recommandé).

Les noms des constantes doivent être entièrement en majuscule. Le séparateur de mot est le caractère de soulignement (*underscore* : «_»). Par exemple, on écrira `TAUX_TPS`. Voici des exemples de nom de constantes qui ne respectent pas cette convention : `TAUXTPS`, `Taux_Tps`, `taux_tps`, `TauxTps`, etc.

Pour donner un nom symbolique à une constante, on utilise le mot clé `final` en Java :

```
public class TesterConstanteSymbolique {

  public static void main ( String [] arsg ) {

    final double TAUX_TPS = 0.15;

    System.out.print( "Le taux de la TPS est " );
    System.out.println( TAUX_TPS );

  }

}
```

Exécution :

```
Le taux de la TPS est 0.15
```

Une fois la valeur d'une constante est associée à un nom symbolique, on ne peut plus changer cette valeur. Le programme suivant provoque une erreur de compilation au niveau de la 4ième ligne de code :

```
public class Exemple {
  public static void main (String [] args) {
    final int NOMBRE = 20;
    NOMBRE = 5; // erreur de compilation
  }
}
```

3 Les variables

Une variable est un emplacement mémoire identifié par un nom (le nom de la variable) dans lequel on peut placer une valeur d'un type spécifique ou écraser une valeur existante.

3.1 Déclaration

L'instruction suivante donne un ordre à l'ordinateur d'allouer un espace mémoire et de nommer cet espace `var`. L'espace alloué doit pouvoir contenir uniquement des valeurs de type `int`.

```
int var;
```

Déclarer une variable consiste à réserver un espace mémoire qui peut contenir un certain type de valeurs et à indentier cet espace à l'aide d'une étiquette (le nom de la variable). Deux informations permettent de déclarer une variable : son type et son nom.

```
public class Exemple {
  public static void main (String [] args) {
    byte varByte;
    short varShort;
    int varInt;
    long varLong;
    float varFloat;
    double varDouble;
    char varChar;
    String varString;
    boolean varBoolean;
  }
}
```

Notez que la convention de codage recommande de ne pas déclarer plus d'une variable sur la même ligne.

Le nom d'une variable doit être en minuscule hormis les initiales des mots le composant (sauf le premier). Il doit être le plus explicite possible sur son contenu et très court. Les symboles de dollars ($) et le soulignement (_) sont proscrits.

Le programme suivant présente une erreur de compilation car il faut déclarer une variable avant de pouvoir afficher sa valeur.

```
public class Exemple {
  public static void main (String [] args) {
    System.out.println( n ); // erreur de compilation
  }
}
```

3.2 Affectation de valeurs

Une fois réservé, l'emplacement mémoire alloué reste vide jusqu'à ce qu'une valeur y soit placée par l'intermédiaire de l'*affectation*.

Règle de base : On peut affecter une valeur à une variable du même type (on verra plus tard le cas particulier des types `byte` et `short`). Voici un exemple :

```
public class Exemple {
  public static void main (String [] args) {
    // declaration
    int varInt;
    long varLong;
    float varFloat;
    double varDouble;
    char varChar;
    String varString;
    boolean varBoolean;

    // affectations de valeur
    varInt = 5;
    varLong = 5L;
    varFloat = 5f;
    varDouble = 5d;
    varChar = 'a';
    varString = "a";
    varBoolean = true;

    // affichage
    System.out.println( varInt ); // affichage
    System.out.println( varLong );
    System.out.println( varFloat );
    System.out.println( varDouble );
    System.out.println( varChar );
    System.out.println( varString );
    System.out.println( varBoolean );
  }
}
```

On peut affecter une valeur initiale à une variable au moment de sa déclaration comme on peut lui affecter une valeur lue au clavier (sauf pour le type `boolean`) :

```
public class Exemple {
  public static void main (String [] args) {
    // declaration et initialisation
    int varInt = 5;
    long varLong = 5L;
    float varFloat = 5f;
    double varDouble = 5d;
    char varChar = 'a';
    String varString = "a";
    boolean varBoolean = true;

    // affichage
    System.out.println( varInt );
    System.out.println( varLong );
    System.out.println( varFloat );
    System.out.println( varDouble );
    System.out.println( varChar );
    System.out.println( varString );
    System.out.println( varBoolean );

    // affectations de valeur
    varInt = Clavier.lireInt();
    varLong = Clavier.lireLong();
    varFloat = Clavier.lireFloat();
    varDouble = Clavier.lireDouble();
    varChar = Clavier.lireCharLn();
    varString = Clavier.lireString();

    // affichage
    System.out.println( varInt );
    System.out.println( varLong );
    System.out.println( varFloat );
    System.out.println( varDouble );
    System.out.println( varChar );
    System.out.println( varString );
  }
}
```

Le programme suivant présente une erreur de compilation car il faut déclarer une variable avant de pouvoir lui affecter une valeur.

```
public class Exemple {
  public static void main (String [] args) {
    n = 5; // erreur de compilation
  }
}
```

Le programme suivant présente une erreur de compilation car il faut placer une valeur initiale dans une variable avant de pouvoir afficher son contenu.

```
public class Exemple {
  public static void main (String [] args) {
    int n;
    System.out.println( n ); // erreur de compilation
  }
}
```

On peut écraser la valeur d'une variable :

```
public class Exemple {
  public static void main (String [] args) {
    int n; // déclaration
    n = 5; // affectation d'une valeur initiale
    System.out.println( n );
    n = 7; // on écrase l'ancienne valeur avec une nouvelle
    System.out.println( n );
  }
}
```

4 DecimalFormat

Pour afficher une variable de type `double` ou `float`, on va utiliser les classes `DecimalFormat` et `DecimalFormatSymbols`.

Étant donné une vaiable `var` de type `double` ou `float`, voici les étapes que vous devez suivre pour afficher cette variable :

1. Importer les classes `DecimalFormat` et `DecimalFormatSymbols` :
   Il s'agit d'écrire la ligne suivante avant la définition de la classe contenant votre programme principal :

   ```
   import java.text.*;
   ```

2. Créer un objet de type `DecimalFormatSymbols` en faisant appel au constructeur `DecimalFormatSymbols()` :
   Au début de la méthode `main`, vous devez avoir une ligne avec l'instruction suivante :

   ```
   DecimalFormatSymbols dfs = new DecimalFormatSymbols();
   ```

3. Déterminer les séparateurs des décimales et des milliers :
   Après la création de l'objet de type `DecimalFormatSymbols()`, vous devez avoir la ligne suivante :

   ```
   dfs.setDecimalSeparator('.');
   ```

4. Créer un objet de type `DecimalFormat` en faisant appel au constructeur `DecimalFormat( "0.00", dfs );` :
   Ajoutez la ligne suivante :

   ```
   DecimalFormat df = new DecimalFormat( "0.00", dfs );
   ```

   En passant la chaîne `"0.00"` au constructeur, on précise qu'on souhaite afficher exactement deux chiffres après la virgule. Si on veut afficher trois chiffres après la virgule, il faut indiquer `"0.000"`.

5. Afficher la variable `var` de type `double` ou `float` :
   Il s'agit d'utiliser l'instruction suivante :

   ```
   System.out.println( df.format( var ) );
   ```

Voici un exemple complet d'utilisation de la classe `DecimalFormat` :

```
import java.text.*;

public class TestDecimalFormat {
    public static void main( String[] args ) {
        DecimalFormatSymbols dfs = new DecimalFormatSymbols();
        dfs.setDecimalSeparator('.');
        DecimalFormat df = new DecimalFormat( "0.00", dfs );

        double d1 = 1.2300000000001;
        double d2 = 1.2999999999999;
        System.out.println( df.format( d1 ) );
        System.out.println( df.format( d2 ) );
    }
}
```

Voici l'affichage obtenu après exécution de ce programme :

```
1.23
1.30
```

5 Références

— DELANNOY, C – Programmer en Java (5ème édition, Java 5 et 6)
Chapitre 2 : Généralités
— DELANNOY, C – Programmer en Java (5ème édition, Java 5 et 6)
Chapitre 3 : Les types primitifs de Java
— `http://www.loria.fr/~tombre/DepInfo/PolyTC/tcinfo004.html`