Rappel cours

Un programme java est constitué d'un ensemble de classes et d'objets, instances de ces classes.
Les objets communiquent par envoi de message.
Un programme doit avoir une classe comportant la méthode *main* qui est le point de départ de l'exécution.
Cette méthode doit impérativement avoir le profil suivant :

```
public static void main(String[] args)
```

Définition d'une classe

Une classe correspond à un type de données défini par le programmeur ou provenant de l'API. Elle peut être manipulée comme un type de base du langage. Elle décrit les propriétés d'un ensemble d'objets

- au niveau structurel : les données (ou champs ou attributs)
- au niveau comportemental : les méthodes

La syntaxe pour définir une classe est la suivante :

```
[public] [final] [abstract] class Nom-classe [spécif-héritage] {
   [liste-champs]          //ordre non impératif
   [liste-constructeur]
   [liste-méthode]
}

spécif-héritage :
[extends Classe-parente] [implements Interface1, …, InterfaceN]]

liste-champs :
0 ou plusieurs déclarations de la forme :
[public|private|protected] [static] [final] type-champ nom-champ [= valeur] ;

liste-constructeur :
0 ou plusieurs définition de constructeur de la forme :
[public|private|protected] Nom-classe ([liste-paramètres]) {
   [liste-instructions]
}

liste-méthode :
0 ou plusieurs définition de méthodes de la forme :
[public|private|protected] [static] [final] type-retour nom-méthode ([liste-
paramètres]) { [liste-instructions] }
```

La convention utilisée pour la syntaxe est :

- les mots clés sont écrits en gras et doivent être recopiés tels quels dans la commande sans modification
- les autres mots doivent être remplacés . Exemple : Nom-classe → à cet endroit de la commande il faut donner l'identificateur de la classe.
- // : commentaire
- [] : délimitent une partie facultative de l'instruction.
- | : choix entre plusieurs options

Important : les symboles [,] et | sont à interpréter. Ils ne figurent pas dans la commande.

Remarques :

- Apparemment, les modificateurs (`public`, …, `static` et `final`) peuvent être permutés.
- Pour les constructeurs, il est impératif que l'identificateur du constructeur soit le nom de la classe. Pas de type-retour. Ne pas confondre avec type-retour = void.
- Les types sont soit des types primitifs, soit des noms de classes pré-définies, soit des noms de classes de l'application.
- Les exceptions ne sont pas prises en compte.

Instructions

Les instructions comportent de nombreuses similitudes avec le langage C.
Quelques différences :

- Il est autorisé de déclarer des variables après des définitions de méthode. La déclaration doit toujours précéder l'apparition de la variable dans une instruction.
- type `boolean`
  → si `i` est de type `int`, ne plus écrire : `if(i) …` mais `if(i!=0)`
  → déclarer `boolean test` et écrire : `if(test)…`
- les tableaux qui ont davantage un comportement objet que de type primitif.
  - déclaration : pas de borne obligatoire. Exemple : `int i[];` ou `int []i;`
  - allocation :
    - `i = new int[100];`
    - `int [] i = {3, 4, 6, 10}; //` idem C
  - accès à un élément (idem C) : `i[3]`
  - taille du tableau : `i.length`

Appel à une méthode :

1. A l'intérieur de la classe où elle est définie : pas de préfixage : `nom-méthode(arguments);`
2. A l'extérieur de la classe où elle est définie :

- pour une méthode d'instance : `nom-objet.nom-méthode(arguments);`
- pour une méthode de classe : N`om-classe.nom-méthode(arguments);`
  // la méthode a été déclarée `static`

Même principe pour les attributs.

Création d'une nouvelle instance de la classe MaClasse ;
`MaClasse classe = new MaClasse(…) ;` // → appel au constructeur approprié.

Conventions d'écriture pour les identificateurs :

- paquetage : nom entièrement en minuscules. Exemple : `java.util`
- classe : son 1er caractère est en majuscule. Les autres caractères sont ne minuscules sauf s'il faut distinguer différents noms dans l'identificateur. Exemple : `class MaClasse`
- attribut ou méthode : 1er caractère de son nom est en minuscule. Les autres caractères sont ne minuscules sauf s'il faut distinguer différents noms dans l'identificateur. Exemples : `abscisse, getAbscisse`
- Constante (définition avec le mot clé `final`) : tous les caractères sont en majuscules. Séparation de deux mots avec underscore Exemple PI, VALEUR_MAX.

Il n'y aura pas d'erreur de compilation ou d'exécution si ces conventions ne sont pas respectées. Elles sont néanmoins vivement recommandées pour une bonne lecture (et correction) du code.

Entrées/sorties

- package `java.io` utilisé
  à importer si utilisation des procédures de façon non triviale : `import java.io.*`
- 3 flux prédéfinis pour les E/S sur des terminaux :
  `System.in` : entrée standard
  `System.out` : sortie standard
  `System.err` : erreur standard

Écriture sur la sortie standard :

objet : `System.out`, méthodes : `print()` ou `println()`
argument : objet de type `String`

- par appel de la méthode `toString()`, appel automatique dans bien des cas
- `\n` autorisé dans la chaîne de caractères : passage à la ligne
- `argument = chaîne + var` est autorisé
  `+` : symbole de concaténation de chaînes de caractères
  `var` est converti en chaîne de caractères (String).
  Exemple :

```
i=3 ;
System.out.println("C'est la " + i + "ème édition.");
```

Lecture d'une fenêtre console
Deux solutions sont possibles :

- arguments transmis sur la ligne de commande (idem C)
  Appel : `java <programme> <args[0]> <args[1]> …`
  Dans le programme java, par exemple : `String ligne=args[0];`
- lecture en cours d'exécution
  objet : `System.in`, méthode : `readLine()`
  `import java.io.*, throws IOException` indispensables pour obtenir une compilation du programme.

```
import java.io.* ;
class … {
      public static void main (String []args) throws IOException {
            String line ;
            BufferedReader br ;
            br=new BufferedReader(new InputStreamReader(System.in));
            System.out.println(«Donner un texte : ») ;
            line=br.readLine() ;
                  …
            }
      }
```

Dans cet exemple, le programme s'arrête s'il y a erreur de lecture.

Dans les deux types de lecture, la donnée lue est de type `String`. Si elle doit être affectée à une variable d'un autre type, il faut exécuter une conversion :

```
String → double :
      double x=Double.parseDouble(line) ;
String → int
      int n=Integer.parseInt(line) ;
```

```
String → char
      int position= … ;
      char c= line.charAt(position)
```

Ces méthodes sont définies de la façon suivante :

- classe `Double` : `public static double parseDouble(String s)`
- classe `Integer` : `public static int parseInt(String s)`
- classe `String` : `public char charAt(int index)`

Programmation java

- saisie du code source sous éditeur
  C'est un ensemble de fichiers : de préférence une seule classe par fichier.
  Si plusieurs classes dans un même fichier, une seule peut être déclarée `public`.
  Le fichier sera sauvegardé avec un nom
  `nomFichier`= `nomClasse`**`.java`**
- compilation : **`javac`** `nomClasse`**`.java`**
  → `nomClasse`**`.class`** en Java Byte Code (JBC)
  autant de `.class` qu'il y a de classes
- exécution : **`java`** `nomClass`
  Ne pas donner d'extension lors de l'exécution.