Programmation - Java - Fiches du Langage

Marie Beurton-Aimar

Fiche Numéro 1

1.1 Structure d'un programme Java

- Toutes les instructions se terminent par un `;`
- Le nom du fichier qui contient une classe doit être le même que celui de la classe.
- Le nom des classes commence par une majuscule, le nom des variables et des méthodes par une minuscule.
- **NB** : toute variable doit faire l'objet d'une déclaration avant d'être utilisée.

1.1.1 Les blocs

- Les caractères de début et de fin de blocs sont respectivement { et } .
- Un fichier contenant du code java doit commencer par l'instruction `class NomDeLaClasse` suivi d'une { la dernière ligne du fichier doit contenir la } qui correspond et ainsi refermer le bloc de définition de la classe.
- Une variable est visible dans son bloc de définition et dans les blocs inclus dans ce bloc.
- Tous les éléments d'un programme doivent se trouver dans une classe.

1.1.2 Les commentaires

- Un commentaire d'une seule ligne peut être signalé par les caractères : `//`
- Un commentaire qui s'étend de 1 à n lignes est encadré par les caractères : `/*` pour marquer le début et `*/` pour la fin.
- Des commentaires peuvent être rédigés pour être intégrés dans le système de documentation du langage : `javadoc`. Pour une description de cet outil se reporter à la section suivante.

1.1.3 La documentation

- `javadoc` est un système de documentation intégré fourni par le langage.
- Définition d'un commentaire pour `javadoc` :
  - Début : `/**`
  - Fin : `*/`
  - Chaque ligne doit commencer par une `*`
  - Le caractère `@` permet de renseigner des champs prédéfinis par `javadoc`. Exemple :

```
/**
 *  Programme de gestion d'expériences
 *  @author  Paul Géranium
 *  @version 1.0
```

- Il est également possible de construire la documentation d'une méthode. Exemple ;

```
/**
 *  Recherche des éléments supérieurs à la moyenne
 *  @param poids moyen de référence
 *  @param liste des éléments
 *  @return la liste des éléments dont le poids est supérieur à la moyenne
 *  @see ClasseCalcul
 */
```

- La commande `javadoc MonProgramme.java` génèrera un fichier `MonProgramme.html` au format HTML et ayant la même présentation que la documentation de l'API java : fichier d'index, présentation de la classe et des méthodes associées, liens dynamiques pour la hiérarchie.
- Attention, si vous désirez générer la documentation complète de votre classe (cad y compris les éléments privés) vous devez utiliser l'option `-private` avec `javadoc`, sinon seul les éléments publics seront documentés. Pour une explications de la notion de public/privé cf section(1.2)

1.2 Paquetage et portée des classes

Un système de visibilité des variables, méthodes et classes est mis en place (indépendemment des notions de blocs vues précédemment) dans les langages objets au moyen de mots clés qui apparaissent lors de la déclaration.

1.2.1 Visibilité des classes et de leur contenu

- Les mots clés `public`, `protected` et `private` permettent de définir la portée des membres d'une classe : champs ou méthodes.
- Interprétation des mots clés lorsqu'ils qualifient un champs ou une méthode :
  - `private` : visible dans la classe.
  - `public` : visible partout - si la classe est elle-même visible.
  - `protected` : visible dans la classe et les sous-classes.
  - Si aucun mot clé n'est spécifié alors la visibilité est la même que celle de la classe.
- Interprétation des mots clés lorsqu'ils qualifient une classe :
  - `public` : visible partout (dans le respect du PATH)
  - `private` : les champs et les méthodes ne peuvent être transmis aux sous classes.
  - Si aucun mot clé n'est précisé la portée de la classe est le paquetage par défaut - le plus souvent le répertoire dans lequel se trouve cette classe.

Fiche Numéro 2

2.1 La classe String

Le langage Java fournit une classe `String` pour gérer les chaines de caractères. Toutes les variables de type `String` sont des constantes. Ceci implique que toute application de méthodes de transformation de la chaine de caractères renvoie une nouvelle chaine et laisse l'ancienne inchangée.

2.1.1 Création

- Par une valeur constante : `String nom=''toto''`
- Appel au constructeur :
  - `String nom=new String()` crée une chaine vide,
  - `String nom=new String(``titi'')` crée une chaine contenant ``titi''
- A partir d'une autre variable : utilisation de la méthode `toString`
  - variable de type `int` : `String motI=Integer.toString(2);`
  - variable de type `doubleString motD=Double.toString(2.0);`
- La méthode `print` (`println`) utilise la méthode `toString`, l'implémentation de cette méthode dans une classe permet d'utiliser ensuite `print` (`println`) pour l'écriture d'information sur les objets de cette classe.

2.1.2 Manipulation - conversion

- La méthode `length` permet de connaitre la taille d'une chaine de caractères. Exemple :

```
String lavariable=''une nouvelle chaine de caracteres'';
System.out.println(``la variable a une longueur''+ lavariable.length());
```

- La méthode `equals` permet de comparer deux chaines de caractères, cette méthode renvoie `true` si les deux chaines sont égales, `false` sinon.
- La méthode `compareTo` permet également de comparer deux chaines de caractères, cette méthode effectue une comparaison lexicographique et renvoie 0 si les deux chaines sont exactement égales, une valeur négative si la chaine qui appelle la méthode est plus petite que celle passée en argument et une valeur positive si elle est plus grande.
- Les méthodes `toUpperCase` et `toLowerCase` renvoient respectivement en majuscule, minuscule, la chaine de caractères initiale.
- La méthodes `valueOf` renvoie aussi un `String` correspondant au paramètre. Exemple :

```
String motI=String.valueOf(1);
String motF=String.valueOf(2.0);
boolean tag=true;
String motB=String.valueOf(tag);
```

2.1.3 la classe StringBuffer

La classe `StringBuffer` permet de créer des chaines qui peuvent être modifiées.

- la méthode `append` permet de modifier la chaine par ajout d'éléments. Exemple:

```
StringBuffer chaine;
chaine = new StringBuffer(‘‘bon’’);
chaine.append(‘‘jour’’);
```

- Les méthodes `length` et `equals` sont disponibles pour cette classe.
- **Attention** : le test `==` n'est pas utilisable pour les objets de cette classe.
- La méthode `toString` permet de récupérer un `String` correspondant à l'objet.

Fiche Numéro 3

3.2 Tests booléens

3.2.1 IF

- Règle d'écriture :

```
if (test)
 {
    instructions si vrai;
 }
else
  {
    instructions si faux;
  }
```

- La partie `else` est optionnelle.
- Il et possible d'imbriquer autant de `if` que désiré.
- Exemple :

```
if (a%2==0)
 {
   System.out.println(a+''est pair'');
 }
else
 {
   System.out.println(a+''est impair'');
 }
```

3.2.2 Switch Case

- Règle d'écriture :

```
switch LaVariableATester
  {
   case unevaleur : instructions si egale;break;
   case uneautrevaleur : instructions si egale;break;
   default : instructions si aucun cas vrai;
  }
```

- Si on ommet `break` le fait d'entrer dans un `case` provoque l'exécution des suivants

- `default` est exécuté si aucun `case` n'est vrai.
- Exemple :

```
switch num
  {
   case 1 :
   case 2 : System.out.println(num+''est egal a 1 ou 2'');break;
   case 3 :System.out.println(num+''est egal a 3'');break;
   default :System.out.println(num + ''est différent de 1, 2, ou 3'');
  }
```

3.3 Boucles et itérations

3.3.1 While

- Règle d'écriture :

```
while(test)
  {
    instructions tant que test est vrai;
  }
```

- Exemple :

```
cpt=0
while (cpt<10)
 {
   System.out.println(cpt);
   cpt++;
 }
```

3.3.2 For

- Règle d'écriture : for(initialisation;test;incrementation) instructions tant que test est vrai;

- Exemple :

```
for (cpt=0;cpt<10;cpt++)
 {
   System.out.println(cpt);
 }
```

Fiche Numéro 4

4.2 Méthodes pour lire sur l'entrée standard

4.2.1 Lire une chaine de caractères

```
   public static String saisie_chaine ()
    {
      try {
        BufferedReader buff = new BufferedReader
            (new InputStreamReader(System.in));
        String chaine=buff.readLine();
        return chaine;
      }
      catch(IOException e) {
System.out.println(" impossible de travailler" +e);
return null;
      }
    }
```

4.2.2 Lire un nombre

```
   public static int saisie_entier ()
    {
       try{
         BufferedReader buff = new BufferedReader
            (new InputStreamReader(System.in));
        String chaine=buff.readLine();
        int num = Integer.parseInt(chaine);
        return num;
       }
      catch(IOException e){return 0;}
    }
```

Fiche Numéro 5

5.2 Les variables et les méthodes de classe

Il est possible de définir des caractéristiques et des comportements qui soient liés aux classes et non aux instances des classes.

- Le mot clé `static` permet de spécifier qu'une méthode ou un attribut est attaché à la classe.
- L'accès à un attribut ou une méthode `static` se fait en citant le nom de la classe. Exemple :

```
class Animal{
   public static int compteur=10;
   public void affiche(){
       System.out.println(‘‘le compteur d’animal est’’+Animal.compteur);
    }
}
```

- Remarque : Le clavier et l'écran sont 2 variables `static` de la classe `System`.
- la conséquence de l'application du qualificatif `static` est qu'il n'existe qu'un exemplaire d'une variable de classe er non un exemplaire par instance.

Fiche Numéro 6

6.2 Classes abstraites et interface

6.2.1 Classes abstraites

- Le mot clé `abstract` déclare une méthode ou une classe abstraite.
- Une méthode abstraite est une méthode dont on fournit la déclaration mais pas l'implémentation (c-à-d le code).
- Toute classe ayant au moins une méthode abstraite devient abstraite et doit être déclarée comme telle.
- Il est interdit de créer une instance de classe abstraite - pas de `new` - mais il est possible de déclarer et manipuler des objets du type de telle classe.

Exemple : les classes `Reader` et `Writer` sont abstraites. Leur rôle est de définir des méthodes génériques de lecture - écriture qui sont ensuite implémentées en fonction du type de données lues : `Buffered, CharArray, FilterReader, InputStream, String`

6.2.2 Utilisation des Interfaces

- Une interface ne contient que des méthodes qui sont par définition abstraites.
- Elle permet le partage de comportements entre des classes qui n'ont pas de lien hiérarchique.
- Toute classe qui déclare implémenter une interface doit fournir le code correspondant aux méthodes de cette interface.
- Il existe un arbre d'héritage entre les interfaces du langage Java.
- La liste des interfaces d'un package est donné au début de la documentation du package.

Fiche Numéro 7

7.2 Exceptions

- Les exceptions servent à gérer les erreurs d'exécution qui peuvent survenir pendant le déroulement d'un programme.
- Chaque classe du langage décrit les exceptions qui peuvent être *levées*.
- Le bloc d'instruction :
  `try {....} catch (TypeException except) {..}`
  permet la capture et le traitement d'une exception.
- Lorsqu'une méthode délègue le traitement d'une exception elle le déclare dans son entête avec le mot clé `throws`.
- Pour lever une exception, on utilise la fonction `throw`
- Exemple :

```
public class LectureException extends IOException{

  LectureException() {/*constructeur vide */}
  public LectureException(String message) {super (message);}
}

public class Essai{

 public static double lireDouble() throws LectureException{
     String ch='' '' ;
     BufferedReader flux =new BufferedREader(new InputStreamReader(System.in));
     try{
         ch=flux.readLine();
         double x=Double.valueOf(ch).doubleValue();
         return x;
        }
     catch (Exception exp){
         throw new LectureException(``Erreur lecture'');
        }
}
```

Fiche Numéro 8

8.1 Entrées - Sorties

8.1.1 Lecture de fichier

```
public static void lire (Vector leclub)throws IOException
{
 BufferedReader buff=new BufferedReader(new FileReader("fichier.txt"));
 try {
   Animal courant=null ;
   for(;;){
      String nom = buff.readLine();
      courant = (Animal) new Animal(nom);
      int num = Integer.valueOf(buff.readLine()).intValue();
      courant.setAge(num);
      String en_vie = buff.readLine();
      if (en_vie.equals("mort")) courant.mourrir();
      leclub.addElement(courant);
     }
  }
 catch (InstantiationException e){
   System.out.println("Fini");
   buff.close();
  }
}
```

8.1.2 Ecriture dans un fichier

```
// Dans le fichier Animalerie
public static void ecrire (Vector leclub)
       throws IOException
{
  BufferedWriter buff=new BufferedWriter
     (new FileWriter("fichier.txt"));
  for(Enumeration e = leclub.elements();e.hasMoreElements();)
    {
     Animal courant = (Animal)e.nextElement();
     courant.save(buff);
    }
   buff.flush();
   buff.close();
}


//Dans le fichier Animal
void save(BufferedWriter buff)throws IOException
 {
    buff.write(nom);
    buff.newLine();
    buff.write((new Integer(age)).toString());
    buff.newLine();
    if (vivant) buff.write("vivant");
    else
       buff.write("mort");
    buff.newLine();
 }
```

8.2 Nomenclature des flux

- Sens du flux : `Reader` et `Writer`
- Type de la source ou de la destination :

Source ou destination	Préfixe du nom de flux
Tableau de caractères	CharArray
Flux d'octets	Input Stream ou OutputStream
Chaine de caractères	String
Programme	Pipe
Fichier	File
Tableau d'octets	ByteArray
Objet	Object

8.3 Flux séquentiels et traitements de données

Traitement	Préfixe
tampon	Buffered
concaténation de flux d'entréee	Sequence
conversion de données	Data
numérotation des lignes	LineNumber
lecture avec retour arrière	PushBack
impression	Print
sérialisation	Object
conversion octets - caractères	InputStream OutputStream

Fiche Numéro 9

9.2 Sauvegarde des objets dans un fichier

9.2.1 Ecriture des objets

- Utilisation de la méthode `writeObject`
  - Cette méthode appartient à la classe `ObjectOutputStream` et prend en paramètre l'objet à écrire sur le flux.
- Exemple de code :

```
public static void sauverObjet(){
    try{
          System.out.println("Donnez le nom du fichier");
          String chaine=saisieChaine();
          FileOutputStream ostream = new FileOutputStream(chaine);
          ObjectOutputStream p = new ObjectOutputStream(ostream);
          p.writeObject(uneAnimalerie);
          p.flush();
          p.close();
    }
    catch (IOException e){System.out.println("Erreur");}
}
```

9.2.2 Lecture des objets

- Utilisation de la méthode `readObject`
  - Cette méthode appartient à la classe `ObjectInputStream` et prend en paramètre l'objet à écrire sur le flux.
- Exemple de code :

```
public static void restaurerObjet(){
   try{
          System.out.println("Donnez le nom du fichier");
          String chaine=saisieChaine();
          FileInputStream istream = new FileInputStream(chaine);
          ObjectInputStream p = new ObjectInputStream(istream);
          uneAnimalerie = (Vector) p.readObject();
          p.close();
       }
   catch (IOException e){System.out.println("Erreur" +e);}

   catch(ClassNotFoundException c){System.out.println("Erreur de chargement");}
}
```

9.2.3 Contraintes

- La lecture et l'écriture des objets sont réalisées grâce à un mécanisme de serialisation. Toutes classes faisant appel à `readObject` et à `writeObject` doit implémenter `Serializable`.
- `Serializable` est une interface sans méthode à implémenter
  - Un objet serialisable est transformable en une suite séquentiel d'octet et inversement
  - donc peut être stocké dans un fichier.
- La sérialisation, c'est lire/écrire un objet dans un flot (flux)
  - permet d'échanger des données entre applications distribuées,
  - permet la persistance des objets
  - un objet est persistant si sa durée de vie est supérieure au programme qui l'a crée.
  - la persistance s'obtient par la sérialisation dans des fichiers
  - on utilise les classes `FileInputStream` et `FileOutputStream` qui sont des flots (flux) pour la lecture/écriture des octets et instancient un flot à partir d'un objet `File`