Chapitre 5

Structure et syntaxe d'une classe JAVA

Ce chapitre décrit la syntaxe du langage JAVA : déclarations des attributs, les types de données, des structures de contrôles, …

Le minimum permettant de commencer à faire de la programmation objet en JAVA.

1. La structure d'une classe

La classe est une déclaration d'un certain nombre d'élément qui appartiennent tous à la classe. Si la classe contient une méthode main alors elle est le début d'exécution de votre programme.

Une classe est composée de deux familles d'éléments :
- les attributs qui sont les données de la classe
- les méthodes qui sont les traitements de la classe
(- classes internes)

Rappels :
Toute classe public est **unique** dans un fichier source .java.
Le nom du fichier doit impérativement du même nom que la classe public.
Un fichier .java peut contenir d'autres classes mais alors ces classes sont toutes des classes dites privées.
L'élément de compilation est le fichier .java.
Le résultat de la compilation d'un fichier .java est la création d'autant de fichier .class qu'il existe de classe dans le fichier .java (la public et les privées).

Structure syntaxique d'une classe :

```
// La classe public

public class <nom de la classe>
{
    //déclarations des attributs de la classe

    //déclarations des méthodes de la classe
}


// Les classes privées de la classe

class ClassePrivee1
{

}

class ClassePrivee2
{

}
```

Comme nous le verrons dans le chapitre 5, chaque classe permet de créer des objets à la demande. On dit que ces objets sont **des instances de la classe**.

Ainsi, au sein d'un programme objet se dessine des ensembles de catégories d'objet en fonction de leurs classes d'appartenance.

Il est évident que chaque objet a ses propres attributs. Cela signifie que lors de la création d'un objet tous les attributs prennent des valeurs par défaut et/ou des valeurs spécifiques.
On appelle ces attributs des **attributs d'objet** (à opposer à attribut de classe que nous verrons un peu plus loin).

2. Les attributs d'objet d'une classe

Une classe est la définition d'un objet au sens donnée informatique.

Les attributs sont avant tout les caractéristiques de l'objet.

Chaque attribut est défini par un **type de donnée** (type au sens large).
Java est un langage fortement typé.

En java, la notion de type de donnée est :
- tous les types **primitifs** (int, long, float, double, char, byte, boolean)
- toutes les **classes** (prédéfinies et non prédéfinies)
- toutes les formes de **tableaux** basés sur un type primitif ou sur une classe

Exemples de déclaration d'attributs:

```
public class Exemple
{
     int nb;
     double x,y;
     int[]   tabEntier;
     String  nom;
}
```

Exemples de classes:

Définition de la donnée pouvant être utilisé dans un programme de gestion d'un Repertoire.

```
public class Individu
{
    public String nom;
    public String prenom;
    public String adresse;
    public String telephone;
}
```

Définition d'un vecteur mathématique utilisé pour construire une bibliothèque de gestion de vecteur dans une espace à deux dimensions.

```
public class Vecteur
{
    public double x;
    public double y;
}
```

Définition d'une équation polynomiale de degré 2.

```
public class Equation2Degre
{
     public double coeffA;  // Coefficient A (entrée)
     public double coeffB;  // Coefficient B (entrée)
     public double coeffC;  // Coefficient C (entrée)

     public double racine1;  // Solution 1 ou unique (résultat)
     public double racine2;  // Solution 2 ou unique (résultat)
     public boolean estSolutionUnique;  // Indique s'il ya qu'1 solution
     public String erreur;
}
```

Définition d'une devise monétaire

```
public class Devise
{
     public String nomDevise1;  // Nom de la devise 1 (ex: FRANC)
     public String nomDevise2;  // Nom de la devise 2 (ex: EURO)
     public double taux;        // Taux de conversion pour passer de
                                //   la devise 1 à la devise 2
}
```

.

Comme nous le verrons par la suite plus en détail, les attributs peuvent être publics ou privées.

Public signifie que l'attribut est visible à l'extérieur de la classe donc accessibles en lecture et en écriture.

Privée signifie que l'attribut est caché pour l'extérieur de la classe donc inaccessibles même en lecture.

```
public class Devise
{
     private String nomDevise1;  // Nom de la devise 1 (ex: FRANC)
     private String nomDevise2;  // Nom de la devise 2 (ex: EURO)
     private double taux;        // Taux de conversion pour passer de
                                 //   la devise 1 à la devise 2
}
```

3. Le constructeur

Dans le cours suivant, nous verrons plus précisément le rôle et les caractéristiques du constructeur.

Le rôle du constructeur est d'initialiser les attributs de l'objet. On lui passe en paramètre les valeurs que l'on veut donner aux attributs de l'objet.

Un constructeur est une méthode sans valeur de retour.

```
public class Devise
{
     private String nomDevise1;  // Nom de la devise 1 (ex: FRANC)
     private String nomDevise2;  // Nom de la devise 2 (ex: EURO)
     private double taux;        // Taux de conversion pour passer de
                                //   la devise 1 à la devise 2

     // Le constructeur
     public Devise(String nom1,String nom2,double t)
     {
       nomDevise1 = nom1;
       nomDevise2 = nom2;
       taux = t;
     }
}
```

Utilisation du constructeur :

```
Devise d = new Devise("FRANC","EURO",6.559);
```

La syntaxe d'un constructeur est:

```
public <nom de la classe>( <paramètres> )
{
     <code du constructeur>
}
```

4. Les attributs de classe (ou attribut statique)

Les attributs statiques sont des attributs qui sont communs à toutes les instances d'une même classe.

Ils sont donc utilisés pour allouer des données qui sont partagées par tous les objets d'une même classe.

Exemple :

```
public class Personne
{
    private String nom;
    private String prenom;
    private int    age;

    static int majorite = 18;

    public Personne(String a_nom, String a_prenom, int a_age)
    {
        nom    = a_nom;
        prenom = a_prenom;
        age    = a_age;
    }

    public boolean siMajeur()
    {
       if (age >= majorite) return(true);
       else return(false);
    }
}

Ailleurs :

Personne o1,o2;

if (o1.siMajeur()) ……..


Personne.majorite = 21;

o1.majorite ➔ 21
o2.majorite ➔ 21
```

Cas particulier :

On utilise très souvent les attributs statiques pour déclarer les constantes qui sont par définition des valeurs communes à toutes les instances de la classe.

On utilise le mot clef final pour préciser que l''entité ne peut être changée.

Il faut que la constante soit initialisée au moment de sa déclaration.

Exemple :

```
 public class Math
{
```

```
        final static public double PI = 3.141592654;
        final static private int MAX = 3;
}

double x = Math.PI * 2;
```

Il est interdit d'écrire ceci :

```
Math.PI = 2.5;
```

On a l'habitude de mettre en majuscule le nom des constantes.

5. Les méthodes objet d'une classe

5.1. Introduction

Les méthodes correspondent au code de votre classe.

Une méthode est un élément de la classe.

Une méthode peut être public ou privée

Une méthode (public/privé ou static) a accès à :
- les paramètres de la méthode
- les variables locales de la méthode
- les attributs d'objet de la classe (attributs non statics)
- les attributs de classe de la classe (attributs statics)

Les rôles des méthodes sont divers. Conceptuellement, on identifie les rôles suivants :
- des méthodes publics permettant de construire l'objet (nous verrons en détail cela dans le chapitre 5)
- des méthodes publics qui accèdent à la valeur d'un attribut (les getteurs)
- des méthodes publics qui changent la valeur d'un attribut (les setteurs)
- des méthodes publics qui permettent à un utilisateur extérieur à la classe d'exécuter un traitement sur l'objet
- des méthodes privées à la classe qui permettent un meilleur découpage des traitements au sein de la classe et permettent de factoriser le code. Ce sont souvent des méthodes statics avec des paramètres afin d'éviter les effets de bords.

5.2. Syntaxe de déclaration d'une méthode d'objet

```
public <type de retour> <nom de la méthode> ( <paramètres>, .... )
{
   <déclarations des variables locales>

   <code de la méthode>
}

private <type de retour> <nom de la méthode> ( <paramètres> )
{
   <déclarations des variables locales>

   <code de la méthode>
}
```

```
<type de retour>
```

est le type de la valeur retournée par la méthode. Si la méthode ne retourne pas de valeur alors **void**
Remarque : le <type de retour> n'est pas précisé quand la méthode est un constructeur.

```
<paramètres>
```

C'est la liste des paramètres formels de la méthode.
Cette liste peut être vide (il reste donc que les parenthèses ())

```
<déclarations des variables locales>
```

Zone de déclarations des variables locales à la méthode.
(ex: int x; int i;)
En fin de compte, Java autorise de déclarer des variables n'importe où dans le code de la méthode.

```
<code de la méthode>
```

Représente le code proprement dit de la méthode.

5.3. Les attributs d'objet de la classe par rapport aux méthodes

Les attributs sont des "variables globales" pour les méthodes de la classe !!!

Exemple :

Voir sur le site http://jacques.laforgue.free.fr l'exemple **Exemple05_Biblio**
Cet exemple sert d'exemple pour la suite.

La méthode toString() n'a aucun paramètre. Elle utilise tous les attributs de la classe comme ce sont des variables globales à la méthode.

5.4. Nomenclature des attributs

Afin de discerner, les attributs de la classe des variables locales et des paramètres, on utilise souvent une nomenclature adaptée.

On peut systématiquement accéder aux attributs par le mot réservé this.

```
public class Individu
{
    private String nom;
    private String prenom;

    public Individu(String nom, String prenom)
    {
        this.nom = nom;
        this.prenom = prenom;
    }
}
```

On peut aussi utiliser une nomenclature :
- préfixer par a_ tous les paramètres

- préfixer par l_ toutes les variables locales (sauf i, j,k)

```
public class Individu
{
    private String nom;
    private String prenom;

    public Individu(String a_nom, String a_prenom)
    {
        nom = a_nom;
        prenom = a_prenom;
    }
}
```

5.5. Le type de retour de la méthode

5.5.1. Le type void

Le type void signifie que la méthode ne retourne pas de valeur.

Mais on peut utiliser quand même l'instruction return qui est un arrêt immédiat de l'exécution de la méthode.

```
public void afficherTitre()
  {
    if (! sousTitre.equals(""))
      {
        Terminal.ecrireStringln(titre + "(" + sousTitre + ")" );
        return;
      }
    Terminal.ecrireStringln(titre);
  }
```

5.5.2. Un type primitif

Une méthode peut retourner un type primitif int, double, char, float, boolean, byte, long.

Dans ce cas l'instruction return doit retourner une valeur du type de la méthode.

```
public int getTome(){return tome;}
```

Syntaxe du return :
return expression
return (expression)

5.5.3. Une chaîne de caractère

Une méthode peut retourner une chaîne de caractère c'est-à-dire un objet de type String.

```
public String toString()
  {
    String str="";
    str=str+"Ident       : "+ident+"\n";
    str=str+"Titre       : "+titre+"\n";
    str=str+"Sous-titre  : "+sousTitre+"\n";
    str=str+"Tome        : "+tome+"\n";
    str=str+"Auteurs     : ";
    for(String s:auteurs) str=str+s+"   ";
```

```
    str=str+"\n";
    str=str+"genre       : "+genre+"\n";
    return ( str );
  }
```

5.5.4. Un tableau

Une méthode peut retourner un tableau

```
public String[] getAuteurs(){return auteurs;}
```

5.6. Le passage des paramètres

Il est essentiel de pouvoir passer des données en paramètre d'une méthode car c'est le moyen le plus efficace pour que la méthode puisse réaliser un traitement qui retourne un résultat différent en fonction de valeurs en entrée.

On appelle paramètres **réels** les valeurs utilisées par l'appelant.
On appelle paramètres **formels** les valeurs utilisées par l'appelé.

Le passage de paramètre consiste à substituer le paramètre formel par le paramètre réel dans l'ensemble du traitement du sous-programme.

En Java, la substitution se fait en substituant la valeur du paramètre. On parle de passage par valeur.

Cette forme de passage est à opposer au passage par référence qui est utilisé dans d'autres langages informatiques mais qui n'est pas utilisé par le langage Java. Le passage par référence consiste à réaliser une substitution du nom du paramètre. Dans ce cas toute modification réalisée sur le paramètre formel est répercutée sur le paramètre réel.

Dans le passage par valeur, le paramètre réel ne peut jamais être modifié par la méthode.

Attention. Comme on le verra pour le tableau (et les objets en général), un tableau étant un pointeur, il est bien passé par valeur mais cela n'interdit pas de modifier ce que le pointeur pointe et donc le contenu du tableau.

5.6.1. Le passage des types primitifs

Dans le main Exemple05_Biblio

```
    int l_tome = 2;
    livre1.setTome(l_tome);
```

Dans la classe Livre :

```
public void setTome(int a_tome){tome = a_tome;}
```

Commentaires :
- le paramètre réel est l_tome
- le paramètre formel est a_tome
- la valeur de l_tome est donnée à a_tome

- l_tome et a_tome sont de même type, ici int.

Preuve que l'on ne peut pas modifier le paramètre passé en paramètre:

```
public class Test2
{
    public static void main(String a_args[])
    {
        int v = 10;
        proc(v);
        Terminal.ecrireIntln(v);
    }

    static void proc(int p)
    {
        p = 100;
    }
}

java Test2
10
```

Et il est impossible de changer la valeur de v qui est passé en paramètre.
La seule manière est de retourner la valeur et l'affecter en résultat.

```
public class Test2
{
    public static void main(String a_args[])
    {
        int v = 10;
        v=proc();
        Terminal.ecrireIntln(v);
    }

    static int proc()
    {
        return(100);
    }
}

java Test2
100
```

5.6.2. Le passage en paramètre d'une chaîne de caractère

Passer en paramètre une chaîne de caractère se fait tout aussi simplement.

```
public void setTitre(String a_titre){titre = a_titre;}
```

Méthode qui change le titre d'un livre.

Comme il est impossible en Java de modifier le contenu d'une chaîne de caractères, il est impossible de modifier une chaîne de caractères passée en paramètre.

5.6.3. Le passage en paramètre d'un tableau

Il est tout aussi simple de passer un tableau en paramètre d'une méthode.

Dans le main de Exemple05_Biblio

```
String[] l_tab = new String[2];
    l_tab[0] = "Philip Jose Farmer";
    l_tab[1] = "K. Cramer";
    livre1.setAuteurs(l_tab);
```

Dans la classe Livre :

```
public void setAuteurs(String[] a_auteurs){auteurs = a_auteurs;}
```

Mais, il est possible de modifier le contenu du tableau car le tableau est un pointeur.

```
public class Test2
{
    public static void main(String a_args[])
    {
        int[] tab = {1, 5, -12, -7, 56, 4, 23, -5 };

        abstab(tab);

        afficherTabint(tab);
    }

    static void afficherTabint(int[] t)
    {
        for(int i=0;i<t.length;i++)
            Terminal.ecrireStringln(i + " : " + t[i]);
    }

    static void abstab(int[] t)
    {
        for(int i=0;i<t.length;i++)
            if (t[i]<0) t[i] = -t[i];
    }
}

java Test2
0 : 1
1 : 5
2 : 12
3 : 7
4 : 56
5 : 4
6 : 23
7 : 5
```

Par contre, il n'est toujours pas possible de changer la valeur du tableau (et non son contenu).

```
public class Test2
{
    public static void main(String a_args[])
    {
        int[] tab = {1, 5, -12, -7, 56, 4, 23, -5 };

        modifiertab(tab);

        afficherTabint(tab);
    }

    static void afficherTabint(int[] t)
    {
        for(int i=0;i<t.length;i++)
            Terminal.ecrireStringln(i + " : " + t[i]);
    }

    static void modifiertab(int[] t)
    {
        t = new int[5];
    }
```

```
}

java Test2
0 : 1
1 : 5
2 : -12
3 : -7
4 : 56
5 : 4
6 : 23
7 : -5
```

5.6.4. Le passage en paramètre d'un objet

Comme nous le verrons plus précisément dans le prochain cours, on peut passer un objet en paramètre d'une méthode.

```
  static public void afficherLivres(Livre... livres)
  {
    for(Livre l:livres)
      Terminal.ecrireStringln( l.toString() );
  }
```

```
public void setIdent(String a_ident){ident = a_ident;}
```

L'objet passé en paramètre peut être modifié par la méthode.

5.7. Les variables locales de la méthode

Une méthode a besoin de déclarer ses propres variables afin de réaliser son traitement.

Cela se fait tout naturellement en déclarant les variables dans le corps de la méthode.

```
    public String toString()
    {
        String str="";
        str=str+"Ident       : "+ident+"\n";
        str=str+"Titre       : "+titre+"\n";
        str=str+"Sous-titre  : "+sousTitre+"\n";
        str=str+"Tome        : "+tome+"\n";
        str=str+"Auteurs     : ";
        for(String s:auteurs) str=str+s+"   ";
        str=str+"\n";
        str=str+"Lien        : "+lien+"\n";
        return ( str );
    }
```

5.8. La surcharge des méthodes au sein de la classe

On appelle la surcharge d'une méthode au sein de la classe, la possibilité du langage de créer des méthodes de même nom mais différencier par le nombre et/ou le type des paramètres de la méthode.

Exemple :

```
    public void setTitre(String a_titre)
```

```
    {
        titre = a_titre;
       sousTitre = "";
    }

    public void setTitre(String a_titre,String a_sousTitre)
    {
        titre = a_titre;
        sousTitre = a_sousTitre;
    }
}
```

6. Les méthodes de classe (ou méthode statique)

Les méthodes statiques sont des méthodes qui peuvent être utilisées sans instancier un objet et permettent de gérer les attributs statiques.

On accède à une méthode statique en préfixant le nom de la méthode par le nom de la classe d'appartenance.

La première méthode statique est la méthode **main** qui ne peut être que statique car lors de la commande java d'exécution d'un programme, la JVM ne possède encore aucun objet et doit démarrer le programme. Elle appelle alors la méthode *main* de la classe référencée dans la commande.

Dans le main de Exemple05_Biblio :

```
Livre.afficherLivres(livre1,livre2);
```

Dans la classe Livre :

```
static public void afficherLivres(Livre... livres)
  {
    for(Livre l:livres)
      Terminal.ecrireStringln( l.toString() );
  }
```

Il est impossible d'utiliser les attributs non statiques de l'objet dans une méthode statique de la classe.

Par contre à l'inverse, il est possible de faire référence à un attribut statique dans une méthode non statique.

Il n'est pas rare de voir des classes qui ne contiennent que des méthodes statiques. Cela arrive souvent quand on veut rassembler des traitements élémentaires basés sur les types élémentaires.
Un exemple d'une telle classe est la classe Terminal que nous avons utilisée jusqu'à maintenant.

```
import java.io.*;

public class Terminal
{
    public static String lireString() // Lire un String
    {
        String tmp="";
        try {
            tmp = new BufferedReader(new
                      InputStreamReader(System.in)).readLine();
        }
        catch (IOException e)
            {
                System.out.println("Erreur de lecture d'une chaine");
            }
        return tmp;
    }

    public static int lireInt()  // Lire un entier
```

Récupération d'un cas d'erreur d'exécution

Conversion d'une chaîne en entier

```
    {
        int x=0;
        try {
            x=Integer.parseInt(lireString());
        }
        catch (NumberFormatException e) {
            System.out.println("Erreur de lecture d'un entier");
        }
        return x ;
    }

    public  static double lireDouble()  // Lire un double
    {
        double x=0.0;
        try {
            x=Double.valueOf(lireString()).doubleValue();
        }
        catch (NumberFormatException e) {
            System.out.println("Erreur de lecture d'un double");
        }
        return x ;
    }

    public  static char lireChar()  // Lire un caractere
    {
        String tmp=lireString();
        if (tmp.length()==0)
            return '\n';
        else
            {
                return tmp.charAt(0);
            }
    }

    public static void ecrireString(String s) // Afficher un String
    {
        System.out.print(s);
    }

    public static void ecrireStringln(String s) // Afficher un String
    {
        ecrireString(s);
        sautDeLigne();
    }

    public static void ecrireInt(int i)  // Afficher un entier
    {
        ecrireString(""+i);
    }

    public static void ecrireIntln(int i)  // Afficher un entier
    {
        ecrireString(""+i);
        sautDeLigne();
    }

    public static void ecrireBoolean(boolean b)  // Afficher un booléen
    {
        ecrireString(""+b);
    }
```

```
    public static void ecrireBooleanln(boolean b) // Afficher un booléen
    {
        ecrireString(""+b);
        sautDeLigne();
    }

    public  static void ecrireDouble(double d)  // Afficher un double
    {
        ecrireString(""+d);
    }

    public  static void ecrireDoubleln(double d)  // Afficher un double
    {
        ecrireDouble(d);
        sautDeLigne();
    }

    public  static void ecrireChar(char c)  // Afficher un caractere
    {
        ecrireString(""+c);
    }

    public  static void ecrireCharln(char c)  // Afficher un caractere
    {
        ecrireChar(c);
        sautDeLigne();
    }

    public static void sautDeLigne()  // Sauter une ligne
    {
        System.out.println();
    }
}
```

7. Paramètres variables

Depuis la version 1.5 de JAVA, il est possible de créer des méthodes qui acceptent un nombre quelconque de paramètre.

La syntaxe de déclaration d'un tel paramètre est:

```
<type>▪▪▪ <param>
```

Exemple :

```
    static public void afficherLivres( Livre... livres)
    {
        for(Livre l:livres)
            Terminal.ecrireStringln( l.toString() );
    }

Avec :

   afficherLivres(l1,l2,l3) ;
```

Les règles sont :
- ~~L'argument variable est unique~~
- Il doit être placé en dernier dans la liste

- On peut lui substituer un tableau ou une liste d'arguments du type indiqué

8. Les structures de contrôle

8.1. Présentation

Les structures de contrôle sont les formes syntaxiques d'un langage qui déterminent le flot d'exécution des instructions.
Elles permettant ainsi de réaliser les traitements algorithmiques.

Dans la programmation objet on utilise les structures de contrôle dans les méthodes.

8.2. Le bloc d'instruction

Le bloc d'instruction (ou séquence d'instruction) est composé d'une suite d'instruction séparés souvent par un caractère particulier ou le retour à la ligne.

En java un bloc d'instruction commence par une accolade ouvrante et se termine par une accolade fermante. Les instructions sont séparées par un point-virgule.
Il n'est donc pas nécessaire de faire de retours à la ligne mais cela est fortement conseillé.

```
{
  <instruction 1> ; <instruction 2> ;
  <instruction 3> ;
  <instruction 4> ;
  ....
}
```

Les instructions s'exécutent de gauche à droite et de haut en bas.

8.3. Les blocs d'exception

Voir le chapitre de cours consacré aux exceptions

8.4. La condition

ou le "if then" et le "if then else".

En programmation, le test conditionnel est une structure de contrôle algorithmique qui utilise les expressions booléennes (voir plus loin)

Ses syntaxes sont :

```
if ( <expression booléenne> )
   <instructions du if>

if ( <expression booléenne> )
   <instructions du if>
else
   <instructions du else>

<instructions du if> et <instructions du else> sont :
   <1 instruction> ;
   ou
   {
      <instruction 1>;
      <instruction 2>;
      . . .
```

```
    }

  Si l'expression booléenne est true alors le programme exécute les
instructions du if sinon exécute les instructions du else.

  Exemple :

  if ( ( (x == 10) || (x > 100) ) && (! fini) )
     {
        y=200;
     }
  z = 10;

  if (x == 10)
     {
        y=200;
     }
  else
     {
        z = 10;
     }
```

8.5. La boucle for en Java

8.5.1. Définition

La boucle for est une structure de contrôle très répandue dans les langages informatiques évolués.
Son rôle est de réaliser des instructions un certain nombre de fois.

Dans le langage Java, il existe deux formes de la boucle for :
- la forme conditionnelle
- la forme énumérative.

La syntaxe de la boucle for conditionnelle :

```
   for( <initialisation> ; <condition> ; <itération> )
   {
     <instructions> ;
   }
```

<initialisation> :
est un instruction qui est évaluée, une seule fois, à l'initialisation de la structure de contrôle.

<condition> :
est une expression booléenne qui est évaluée, à chaque cycle de l'itération.
Elle est évaluée au début de l'exécution de <instructions>.
Si cette expression booléenne est false alors la boucle est arrêtée.

< itération > :
est une expression qui est évaluée, à chaque cycle de l'itération.
Elle est évaluée à la fin de l'exécution de <instructions>.

Exemple :
Afficher les n premiers nombre entier.

```
public class Exemple
{
    public static void main(String a_args[])
    {
        int i;
        int n;
        n = 10;
        for(i=1; i<=n; i=i+1)
            {
                Terminal.ecrireIntln(i);
            }
        Terminal.sautDeLigne();
    }
}

java Test2
1
2
3
4
5
6
7
8
9
10
```

8.5.2. Les erreurs

ATTENTION aux erreurs que vous pourriez faire dans l'écriture des expressions d'un boucle for.

On peut se trouver avec une boucle infinie.

Exemple : On a oublié la lettre i dans l'expression d'itération

```
for(i=1; i<=n; i=+1)
    {
        Terminal.ecrireString("*");
    }
Terminal.sautDeLigne();
```

Exemple : On a mis i à la place de n dans l'expression booléenne

```
for(i=1; i<=i; i=i+1)
    {
        Terminal.ecrireString("*");
    }
Terminal.sautDeLigne();
```

8.5.3. Exemple

On veut déterminer si un entier supérieur à 2, est un nombre premier.

Rappel : un entier est un nombre premier s'il n'est divisible par aucun entier sauf 1 ou lui-même.

Faire le programme Java qui teste si 5 entiers saisis à l'écran sont des nombres premiers.

En Java, x est divisible par y si le reste de x par y est égale à 0 (x%y == 0).

(le modulo)

```
public class NombrePremier
{
    public static void main(String a_args[])
    {
        int i;
        int n;
        int k;
        int v;
        boolean premier;
        n = 5;
        for(i=1; i<=n; i=i+1)
            {
                v = Terminal.lireInt();
                premier=true;
                for(k=2;k<v;k=k+1)
                    {
                        if (v%k == 0)
                            premier = false;
                    }
                if (premier)
                    Terminal.ecrireStringln("PREMIER");
                else
                    Terminal.ecrireStringln("PAS PREMIER");
            }
    }
}

java Test2
5
PREMIER
2
PREMIER
7
PREMIER
12
PAS PREMIER
14
PAS PREMIER
```

8.5.4. L'instruction break

Cette instruction permet de "casser" une boucle for. C'est-à-dire de sortir immédiatement de la boucle for.

L'exemple précédent boucle jusqu'à ce que k est égal à v-1 malgré que l'on ait trouvé un diviseur, l'instruction break permet de sortir de la boucle une fois que l'on a trouvé un diviseur.

```
 public class NombrePremier
{
    public static void main(String a_args[])
    {
        int i;
        int n;
        int k;
        int v;
        boolean premier;
        n = 5;
```

```
          for(i=1; i<=n; i=i+1)
              {
                  v = Terminal.lireInt();
                  premier=true;
                  for(k=2;k<v;k=k+1)
                      {
                          if (v%k == 0)
                             {
                                premier = false;
                                break;
                             }
                      }
                  if (premier)
                      Terminal.ecrireStringln("PREMIER");
                  else
                      Terminal.ecrireStringln("PAS PREMIER");
              }
      }
  }
```

8.5.5. Simplifications

Afin de simplifier le code et parce que la variable de boucle n'est utilisée que dans la boucle for, on peut déclarer la variable de boucle dans la boucle for :

```
for(int i = 1; i<=n ; i=i+1)
```

On peut aussi simplifier l'incrémentation :

```
i = i +1;
```

est équivalent avec

```
i++;
```

On peut aussi simplifier les blocs qui contiennent qu'une seule instruction.
On peut aussi faire les déclarations de même type dans une même ligne.
On peut aussi déclarer une variable tout en l'affectant.

On obtient alors le programme suivant :

```
public class NombrePremier
{
    public static void main(String a_args[])
    {
        int v;
        boolean premier;
        int n = 5;
        for(int i=1; i<=n; i++)
            {
                v = Terminal.lireInt();
                premier=true;
                for(int k=2;k<v;k++)
                    if (v%k == 0)
                       premier = false;
                if (premier)
                    Terminal.ecrireStringln("PREMIER");
                else
                    Terminal.ecrireStringln("PAS PREMIER");
            }
    }
}
```

8.5.6. La boucle for énumérative

La boucle for dite "énumérative" est utilisé sur une donnée qui possède la propriété dont ses éléments peuvent s'énumérés.

Cette possibilité a été introduite à partir de la version 1.5 de Java.

Les données qui sont énumératives sont par implémentation les tableaux Java et aussi un certain nombre de classe prédéfinies de genre de COLLECTION; Exemples: ArrayList, Vector, …

La syntaxe d'une boucle for énumérative est:

```
for ( <variable> : <donnée> )
   {
     utilisation de <variable> qui prend une valeur différente à chaque tour de boucle
   }
```

Cette nouvelle façon de faire est très répandue et privilégiée dans tous les programmes JAVA.

"La variable est de type de l'élément de la donnée."

Exemple :

```
int[] tab = {12, 23, 4, 6};

for(int x : tab )
    System.out.println(x);

À la place de :

for(int i=0;i<tab.length;i++)
    System.out.println(tab[i]);
```

8.6. La boucle while

8.6.1. Définition

La boucle while est aussi une instruction de contrôle très répandue dans les langages évolués.

Son rôle est de réaliser des instructions tant qu'une certaine condition est vraie.

La syntaxe de la boucle while est :

```
while ( <condition> )
{
  <instructions>
}
```

<condition> :

est une expression booléenne qui est exécutée dès le début et à chaque tour de boucle.

Tant que la condition est true, la boucle continue.

<instructions> :

sont les instructions qui s'exécutent à chaque tour de boucle.

Remarque : une boucle for peut être écrite avec une boucle while :

```
for( <initialisation> ; <condition> ; <itération> )
   <instructions>;

   est équivalent à :

<initialisation>;
while (<condition>)
 {
   <instructions>;
   <itération>;
 }
```

L'instruction break peut être aussi utilisée pour la boucle while.

8.6.2. Exemple

Ecrire l'exemple précédent du test d'un nombre premier avec un while.

```
public class Test2
{
    public static void main(String a_args[])
    {
        // Declaration des variables
        int i;
        int n;
        int k;
        int v;
        boolean premier;
        boolean fini;

        // Nombre de test
        n = 5;
        for(i=1; i<=n; i=i+1)
            {
                v = Terminal.lireInt();
                premier = true;
                fini    = false;
                k=2;
                while (!fini)
                    {
                        if (k>=v)
                            {
                                premier=true;
                                fini=true;
                            }
                        else
                            {
                                if (v%k == 0)
                                    {
                                        premier = false;
                                        fini = true;
                                    }
                                else
                                    {
                                        k=k+1;
                                    }
                            }
                    }
                if (premier)
                    Terminal.ecrireStringln("PREMIER");
                else
                    Terminal.ecrireStringln("PAS PREMIER");
            }
    }
}
```

8.7. La boucle do-while

8.7.1. Définition

La boucle do-while est une boucle qui permet de faire des instructions tant qu'une certaine condition est vraie.

La grande différence avec la boucle while, est que le bloc d'instruction de la boucle est exécuté toujours au moins une fois car la condition d'arrêt est exécutée à la fin du cycle de la boucle.

```
do {
   <instructions>;
} while (<condition>);
```

<condition> :

est une expression booléenne qui est exécutée à la fin de chaque tour de boucle.

Tant que la condition est true, la boucle continue.

<instructions> :

sont les instructions qui s'exécutent à chaque tour de boucle.

8.7.2. Exemple

Saisir une valeur positive et demander la saisie si cela n'est pas le cas :

```
        int v;
        do{
            Terminal.ecrireString("Saisir un entier positif: ");
            v = Terminal.lireInt();
        }while(v<0);
        Terminal.ecrireIntln(v);


java Test2
Saisir un entier positif: -2
Saisir un entier positif: -10
Saisir un entier positif: 3
3
```

Si on remplace l'instruction do-while par un while :

```
        int v;
        while (v<0){
            Terminal.ecrireString("Saisir un entier positif: ");
            v = Terminal.lireInt();
        }
        Terminal.ecrireIntln(v);
```

On obtient l'erreur de compilation suivante:

```
javac Test2.java
Test2.java:9: variable v might not have been initialized
        while (v<0){
               ^
1 error
```

car la variable v n'est pas initialisée !!!

Et il faut que l'on "force" la valeur avec une valeur négative !

```
        int v=-1;
        while (v<0){
            Terminal.ecrireString("Saisir un entier positif: ");
            v = Terminal.lireInt();
        }
        Terminal.ecrireIntln(v);
```

8.8. Le switch case

Le swith case est une structure de contrôle de type conditionnel. Rien à voir avec les boucles et les tableaux.

8.8.1. Définition

Le switch case permet de réaliser des instructions en fonction de différents cas que peut prendre une valeur.

La syntaxe est :

```
switch ( <variable> )
{
   case <constante_1> :
      <instructions 1>;
      break;
   case < constante _2> :
      <instructions 2>;
      break;
   ...
   default :
      <instructions default>;
      break;
}
```

Les instructions break sont optionnelles.
Le cas "default" est optionnel.
Les différents "cas" sont des constantes littérales ou des constantes Java.

8.8.2. Exemple

```
        char valeur;

        Terminal.ecrireString("Saisir un caractère: ");
        valeur = Terminal.lireChar();
        switch (valeur)
            {
            case 'a' :
                Terminal.ecrireStringln("voyelle");
                break;
            case 'e' :
                Terminal.ecrireStringln("voyelle");
                break;
            case 'i' :
                Terminal.ecrireStringln("voyelle");
                break;
            case 'o' :
                Terminal.ecrireStringln("voyelle");
                break;
            case 'u' :
```

```
                Terminal.ecrireStringln("voyelle");
                break;
            case 'y' :
                Terminal.ecrireStringln("voyelle");
                break;
            default:
                Terminal.ecrireStringln("consonne");
                break;
            }
```

Est équivalent à:

```
        char valeur;

        Terminal.ecrireString("Saisir un caractère: ");
        valeur = Terminal.lireChar();
        switch (valeur)
            {
            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u':
            case 'y':
                Terminal.ecrireStringln("voyelle");
                break;
            default:
                Terminal.ecrireStringln("consonne");
                break;
            }
```

On peut utiliser des constantes Java :

```
public class Test2
{
    static final public char lettre_a = 'a';  // Constante Java

    public static void main(String a_args[])
    {
        char valeur;

        Terminal.ecrireString("Saisir un caractère: ");
        valeur = Terminal.lireChar();
        switch (valeur)
            {
            case Test2.lettre_a:
            case 'e':
            case 'i':
            case 'o':
            case 'u':
            case 'y':
                Terminal.ecrireStringln("voyelle");
                Terminal.ecrireStringln("-------");
                break;
            default:
                Terminal.ecrireStringln("consonne");
                break;
            }

    }
```

```
}
```

9. Les types de données élémentaires

9.1. La déclaration d'une variable

La déclaration d'une variable consiste à associer le type de donnée de la variable à son nom symbolique.
On parle aussi de Domaine de définition de la variable.

La déclaration en Java se fait suivant la syntaxe suivante :

```
<type de donnée> <variable>;

<type de donnée> <variable> = <valeur>;

<type de données> <variable1>, <variable2>, <variableN>;
```

La déclaration d'une variable doit se faire avant son utilisation.
Il est possible de déclarer et affecter en même temps.

La déclaration d'une variable en Java peut être faite n'importe où dans le bloc des instructions. Mais, il est fortement conseillé, dans un premier temps, de faire les déclarations en début du bloc des instructions de la méthode main.

Le nom d'une variable est un identificateur, c'est à dire commençant nécessairement par une lettre, majuscule ou minuscule, qui peut être ou non suivie d'autant de caractères que l'on veut parmi l'ensemble [a..z][A..Z][O..9] et les caractère '$' et '_'.

Un nom de variable ne peut pas être un mot réservé du langage (53 mots réservés):

abstract	else	interface	switch
assert	enum	long	synchronized
boolean	extends	native	this
break	false**	new	throw
byte	final	null**	throws
case	finally	package	transient
catch	float	private	true**
char	for	protected	try
class	goto*	public	void
const*	if	return	volatile
continue	implements	short	while
default	import	static	
do	instanceof	strictfp	
double	int	super	

(*) *réservé mais pas utilisé*

(**) réservé mais pas clé

9.2. Les types de données

Il existe de nombreux type de données en Java. Il existe deux grandes familles de types de données :

- les types primitifs
- les types référence (ou objet).

Les types primitifs en Java sont les suivants :

Le type entier

byte: entier codé sur 8 bits (de -128 à 127)
char: caractère unicode sur 16 bits. Sa constante s'écrit: 'a' '\\t' '\\u03a9'
short: entier codé sur 16 bits (de -32768 à 32767)
int: entier codé sur 32 bits
long: entier codé sur 64 bits

Le type flottant

float: nombre en virgule flottante sur 32 bits (simple précision)
double: nombre en virgule flottante sur 64 bits (double précision)

Le type booléen:

boolean, constitué des constantes true et false

Les types références en java sont très nombreux, car toute classe est un type référence. Il est donc impossible de faire une liste exhaustive.

Les tableaux (ex int[]) sont des types références.

Exemples de type référence :

Le type chaîne de caractère :

String correspond à une suite continue d'élément de type primitif char.
Livre correspond à la description d'un livre dans une bibliothèque de prêt.
ArrayList<> collection Java prédéfinie permettant de gérer des tableaux dynamiques.

9.3. Le type énumératif

9.3.1. Introduction

C'est le paradigme de Pascal (type saison = {printemps, ete, automne, hiver}).
Cette facilité n'existait pas en Java. Il fallait définir des constantes entières, c'est-à-dire faire un codage manuel. Pour définir des couleurs par exemple :

public final int ROUGE = 1;
public final int VERT = 2;
public final int BLEU = 3;
Le codage par un int permet la substitution de n'importe quel int pour une couleur.
Ces codes doivent être dans le source du programme pour être accessibles.
Si on imprime la valeur de la variable couleur, on doit décoder...

Cette nouvelle façon de faire est apparue avec la version 1.5 de JAVA.

9.3.2. Définition

Une énumération est un ensemble d'identificateurs affecté à une variable :

enum Couleur = {ROUGE, VERT, BLEU};

- Il s'agit d'un nouveau type java
- Il implémente les interfaces Comparable et Serializable
- Il possède une méthode statique values() qui retourne un tableau de toutes les valeurs dans l'ordre de la déclaration

Le terme "enum" est une sorte de "class".

Tout type enum hérite de la classe java.lang.enum

9.3.3. Exemple

Dans Exemple05_Biblio, la classe GenreLivre :

```
public enum GenreLivre {
  HISTOIRE,
    GEOGRAPIE,
    ROMAN,
    SF
    }
```

```
Livre livre1 = new Livre("12112011-0001",
                         "Les Dieux du fleuve",
                         "Le noir dessein",
                         auteurs,
                         0,
                         GenreLivre.ROMAN);
```

```
```

```
      if (livre2.getGenre().equals(GenreLivre.SF))
        Terminal.ecrireStringln("EGAL");
```

```
  public class Livre
  {
    private String      ident;      // Identification unique du livre
    private String      titre;      // Titre du livre
    private String      sousTitre;  // Sous-titre du livre
    private String[]    auteurs;    // Les auteurs du livre
    private int           tome;       // Tome du livre (0 si pas plusieurs
tome)
    private GenreLivre  genre;      // Genre du livre
  …
  }


    public String toString()
    {
      String str="";
      str=str+"Ident       : "+ident+"\n";
      str=str+"Titre       : "+titre+"\n";
      str=str+"Sous-titre  : "+sousTitre+"\n";
      str=str+"Tome        : "+tome+"\n";
      str=str+"Auteurs     : ";
      for(String s:auteurs) str=str+s+"   ";
      str=str+"\n";
      str=str+"genre       : "+genre+"\n";
      return ( str );
    }
```

Un autre exemple : Exemple36

Voir sur le site http://jacques.laforgue.free.fr l'exemple **Exemple36_Enum**

```
```

9.4. Les expressions

Les expressions sont des calculs qui utilisent et mélangent des :
- constantes
- variables
- opérateurs

Il est important que toutes les constantes et variables utilisées dans une expression soient de même type ou au moins de types compatibles. Nous approfondirons plus tard, cette notion de compatibilité qui passe par une conversion automatique des éléments.

Une expression est donc d'un type précis. Comme le résultat du calcul d'une expression est stocké dans une variable, il faut que cette variable soit du même type.

```
  <T> <variable>;  // déclaration

  <variable> = <expression de type T>;
```

9.4.1. Les expressions de type entier

Une expression entière utilise les opérateurs :

+	addition	ou	opérateur unaire positif
-	soustraction	ou	opérateur unaire négatif
*	multiplication		
/	division entière		
%	modulo entier	(reste de la division entière)	

Exemples:

```
        int v;

v = 10 *3;
System.out.println(""+v);   // 30

int u;
u = 100;
v = u + v;
System.out.println(""+v);   // 130
v = - v / 2;
System.out.println(""+v);   // -65
v = v / -2;
System.out.println(""+v);   // 32

v = 2 + 3 * 4;
System.out.println(""+v);   // 14
v = (2 + 3) * 4;
System.out.println(""+v);   // 20
int w=3;
v = u%( (v/(4+w))+1);
System.out.println(""+v);   // ??
```

```
int val_int = 3;

short val_short = val_int;
```

La compilation de ce code :

```
d:>javac Test1.java
javac Test1.java
Test1.java:28: possible loss of precision
found   : int
required: short
        short val_short = val_int;
                          ^
1 error

Valable :
       short val_short =  (short) val_int;   // Le CAST
```

Cela démontre que Java vérifie à la compilation la compatibilité des types entre eux.

9.4.2. Les expressions de type flottant

Une expression flottante utilise les opérateurs :

+	addition	ou	opérateur unaire positif
-	soustraction	ou	opérateur unaire négatif
*	multiplication		
/	division réelle		

9.4.3. Les expressions de type booléen

Une expression booléenne utilise les opérateurs ;

!	négation logique
&&	ET logique
||	OU logique

Exemples;

```
boolean fini;
boolean trouve;

fini = false;
trouve = false;
int i = 1;
while (! fini)
    {
        if (i > 10 || trouve)
         {
             fini = true;
         }
     }
```

Les opérateurs de comparaisons constituent des expressions booléennes :

>	teste si supérieur
<	teste si inférieur
>=	teste si supérieur ou égale
<=	teste si inférieur ou égale
==	teste si égales
!=	teste si différents

```
if ( (x>10) && (x<100) )   // Teste d'un intervalle

char c;
c = 'x';
if ( c < 'd')    // comparaison de char
```

9.4.4. Les expressions de type chaîne

Création d'une chaîne :

```
String str;

str = "Ceci est un exemple";
```

Concaténation de chaîne : l'opérateur "+"

```
String str1;
String str2;

str1 = "Ceci est un exemple";
str2 = " pas très compliqué";
```

```
str3 = str1 + str2;

int v = 12;
System.out.println("La valeur de v est : " + v);
```

La concaténation d'une chaine avec un entier ou un double est également réalisable simplement :

Les caractères d'une chaine sont accessibles par charAt(i) avec i qui va de 0 à n-1
n étant le nombre de caractère de la chaîne.

Accès à la longueur de la chaîne et l'accès aux caractères :

```
boolean fini;
int i;

fini = false;
i=0;
while (! fini)
    {
        if (i>str3.length()-1)
            fini = true;
        else
            {
                char c = str3.charAt(i);
                System.out.println("Caractère " + i + " : " + c);
                i = i + 1;
            }
    }
```

On peut remplacer le code précédent par :

```
int i=0;
char[]   tab = str3.toCharArray();
for(char c : tab)
 {
     System.out.println("Caractère " + i + " : " + c);
     i++;
}
```