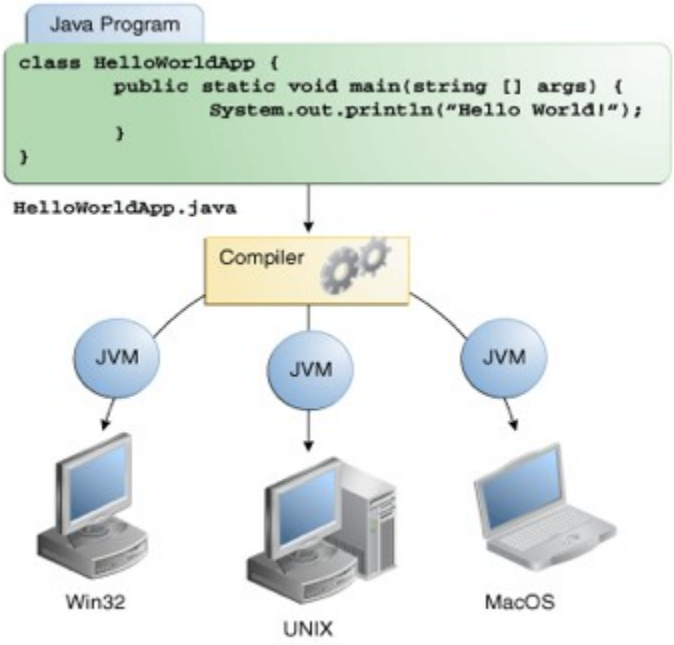
Cours de Java : Les bases Chapitre 1

Ce chapitre présente les connaissances indispensables à l'acquisition des techniques de la programmation en Java : les éléments fondamentaux de la syntaxe du langage ainsi que les concepts de la programmation orientée objet (POO).

Java est un langage de programmation et un environnement d'exécution. Les applications peuvent s'exécuter sur toutes les plates formes (Windows, Linux, Unix, Mac) qui possèdent un environnement d'exécution **JRE** (Java Runtime Environment) adapté. Le site d'Oracle propose les JRE pour les différents systèmes d'exploitation actuels.

Le **JRE** inclut une **JVM**, machine virtuelle Java (Java Virtual Machine), ainsi que les librairies et les composants nécessaires à l'exécution des applications java.

Une application java peut être développée sur une machine Windows, et par exemple s'exécuter sur une machine Linux: on dit qu'une application Java est **portable**.



Le langage Java a été développé par la société Sun. La société Sun n'existe plus car elle a été rachetée par Oracle.
Les exemples suivants seront développés et exécutés sur une machine Windows.
Il existe des produits de développements performants comme **Eclipse** ou **JBuilder** ou **NetBean** que nous n'utiliserons pas dans ce chapitre. Ces produits ne sont pas indispensables, et pour l'apprentissage du langage Java, mieux vaut ne pas les utiliser tout de suite.

1. Outils de base nécessaires au développement d'applications Java.

- Installer un **JDK**, Java Developers Kit, par exemple le jdk 1.6 qui s'installe par défaut dans le dossier **C:\Program Files\Java\jdk1.6.0_25**.
- Dans le **panneau de configuration** de Windows, **Système/Avancé/Variables d'environnement**, ajouter à la **variable système PATH**, le chemin suivant:

C:\Program Files\Java\jdk1.6.0_25\bin.

C'est dans ce répertoire que l'on trouve les commandes nécessaires à la compilation et à l'exécution des applications Java, et notamment:

javac qui est la commande de compilation.

java qui est la commande d'exécution.

- Utiliser un éditeur de texte ASCII comme **Notepad++** pour écrire les programmes sources.

Pour développer une application Java, on passe par les étapes suivantes:

- Ecriture du fichier source, de nom d'extension **.java**, avec un éditeur de texte, enregistrement du fichier.
- Compilation avec le compilateur **javac** dans une fenêtre console (invite de commandes). On utilise pour cela la commande **javac**.
- Exécution avec l'interpréteur **java** dans une fenêtre console. On utilise pour cela la commande **java**.

Tester l'installation

Ouvrir une fenêtre console (invite de commandes ou Démarrer/Exécuter/cmd puis OK). Taper **javac**, une liste de messages propres à Java doit s'afficher.
Si un message du genre « javac n'est pas reconnu comme commande interne ou externe… » est affiché, cela veut dire que soit le jdk n'est pas installé, soit le PATH n'est pas correctement configuré, voir ci-dessus.

2. Premier exemple.

Le langage Java est 100% objet, et toute application est contenue dans une ou plusieurs classes.

- Créer un dossier pour les exercices, par exemple courstp1.
- Ecrire le fichier source ci-dessous, avec Notepad++.

```
public class Bonjour1
{
   public static  void main(String[] args)
   {
    System.out.println("Bonjour !! ");
   }
}
```

- Enregistrer le fichier sous le nom exact: **Bonjour1.java**
  Vous n'avez pas d'autre choix, car java impose l'extension .java, et veut que le nom du fichier soit le même que le nom de la classe, aux majuscules minuscules près.
  bonjour1.java Bonjour1.txt essai.java sont 3 erreurs.

*Il faut maintenant ouvrir une fenêtre console (invite de commandes ou terminal ou exécution de **cmd** sous Windows) et se déplacer avec la commande **cd** dans le dossier contenant le fichier Bonjour1.java. La commande **dir** (**ls** sous Linux) dans ce dossier doit afficher le nom du fichier Bonjour1.java.*

- Compiler le fichier Bonjour1.java avec le compilateur javac, en écrivant la commande suivante :

  javac Bonjour1.java

  S'il y a des erreurs de syntaxe, elles sont indiquées par le compilateur, et il faut revenir dans l'éditeur pour les corriger.
  Si la compilation réussit, elle crée un fichier **Bonjour1.class**.

  Le fichier Bonjour1.class est un fichier de **bytes codes**, ou suite d'octets, qui seront exécutés par l'interpréteur java. **La machine virtuelle java** contrôle l'exécution des bytes codes, comme le fait un microprocesseur qui exécute une suite d'instructions.

- Exécution de l'application avec l'interpréteur java, en écrivant la commande suivante :

  java Bonjour1

  L'exécution affiche **Bonjour !!** sur l'écran en mode texte.

Explication du premier exemple.

- On doit écrire une **classe**. On a choisi le nom **Bonjour1** comme nom de la classe.
- Dans cette classe, il n'existe qu'une seule **méthode**, c'est le point d'entrée du programme, **la méthode main().** Tout programme exécutable java doit contenir une classe avec une méthode **main**.

- Dans la méthode main(), on réalise l'affichage simple d'un texte avec la ligne de code **System.out.println("Bonjour !! ");**

  La classe **System** est une classe du **package java.lang**.
  out permet d'accéder à l'écran qui est la sortie d'affichage standard. **out** est un **objet** (flux instance de la classe PrintStream) qui identifie la sortie standard, c'est objet est contenu dans la classe System, il est systématiquement et automatiquement créé pour chaque exécution d'un programme Java.
  println est une méthode (fonction membre) de la classe **PrintStream** pour écrire dans un flux de sortie, avec retour à la ligne.

  Une application java commence le plus souvent par des **importations** de packages, en écrivant des lignes de code comme
  import java.lang.*;

  java.lang est le seul package importé automatiquement, et il est inutile de l'importer.
  Un package est une entité java qui regroupe plusieurs classes déjà écrites par les développeurs du langage. De nombreux packages existent : pour le réseau, pour faire des ihm (fenêtres), pour accéder aux bases de données...

- **public static void main(String[] args)**

La méthode main() peut recevoir une liste d'arguments de type **String**, elle est **public comme la classe car ainsi elle peut être appelée/utilisée par un objet externe à la classe Bonjour1**, elle est **static car elle doit être appelée par la JVM alors que l'objet instance de la classe Bonjour1 n'est pas créé,** elle ne retourne jamais de valeur et donc est de type **void**.
La JVM exécute **Bonjour1.main()** pour exécuter l'application.

Exercice : Importer le package java.lang dans le premier exemple, compiler, exécuter, qu'est-ce que cela change ?

Pour écrire des applications Java, il faut acquérir de l'expérience sur les packages et les classes, et utiliser la documentation Java Doc accessible sur le site d'Oracle ou téléchargeable.

Les classes du package java.lang.

Class Summary	
Boolean	The Boolean class wraps a value of the primitive type `boolean` in an object.
Byte	The Byte class is the standard wrapper for byte values.
Character	The Character class wraps a value of the primitive type `char` in an object.
Class	Instances of the class `Class` represent classes and interfaces in a running Java application.
Integer	The Integer class wraps a value of the primitive type `int` in an object.
Long	The Long class wraps a value of the primitive type `long` in an object.
Math	The class `Math` contains methods for performing basic numeric operations.
Object	Class `Object` is the root of the class hierarchy.
Runtime	Every Java application has a single instance of class `Runtime` that allows the application to interface with the environment in which the application is running.
Short	The Short class is the standard wrapper for short values.
String	The `String` class represents character strings.
StringBuffer	A string buffer implements a mutable sequence of characters.
System	The `System` class contains several useful class fields and methods.
Thread	A *thread* is a thread of execution in a program.
Throwable	The `Throwable` class is the superclass of all errors and exceptions in the Java language.

Le package java.lang regroupe des classes pour les traitements généraux réalisés par les applications : entrées sorties de données, types de données et conversions, calculs mathématiques. Il n'est pas faux de voir un package comme une librairie de classes.

La classe **System** est utilisée pour les saisies au clavier et les affichages en mode consoles.
La classe **String** permet de manipuler les chaînes de caractères.
La classe **Math** contient de nombreuses méthodes pour effectuer des calculs mathématiques.

Il existe en java des données de type : **char, int, double, long..**.
Il existe également des classes **Character, Integer, Double, Long, Short, Boolean** pour manipuler les types de bases.

La classe **Integer** permet des opérations sur le type **int**, comme des conversions :
Voir ci-dessous les méthodes de la **classe Integer**.

Exemple :
La méthode **parseInt()** de la classe **Integer** convertit une String en int, si cela a un sens:

```
int Nb1;
Nb1=Integer.parseInt("123") ;
```

La méthode **parseInt()** convertit une chaîne de caractères en entier.
Par exemple, si on saisit une valeur numérique au clavier, comme «123», on l'obtient sous forme de **String**. Si on en a besoin comme nombre, on doit la convertir. La variable Nb1 de type int va contenir la valeur convertie numérique 123 que l'on pourra additionner, multiplier etc….

Method Summary	
byte	**byteValue**() Returns the value of this Integer as a byte.
boolean	**equals**(Object obj) Compares this object to the specified object.
int	**hashCode**() Returns a hashcode for this Integer.
int	**intValue**() Returns the value of this Integer as an int.
long	**longValue**() Returns the value of this Integer as a long.
static int	**parseInt**(String s) Parses the string argument as a signed decimal integer.
static int	**parseInt**(String s, int radix) Parses the string argument as a signed integer in the radix specified by the second argument.
short	**shortValue**() Returns the value of this Integer as a short.
static String	**toBinaryString**(int i) Creates a string representation of the integer argument as an unsigned integer in base 2.
static String	**toHexString**(int i) Creates a string representation of the integer argument as an unsigned integer in base 16.
static String	**toOctalString**(int i) Creates a string representation of the integer argument as an unsigned integer in base 8.
String	**toString**() Returns a String object representing this Integer's value.
static String	**toString**(int i) Returns a new String object representing the specified integer.
static String	**toString**(int i, int radix) Creates a string representation of the first argument in the radix specified by the second argument.
static Integer	**valueOf**(String s) Returns a new Integer object initialized to the value of the specified String.
static Integer	**valueOf**(String s, int radix) Returns a new Integer object initialized to the value of the specified String.

3. Deuxième exemple.

Dans cet exemple, l'utilisateur entre au clavier le nombre de fois qu'il dit Bonjour.
Puis, le texte Bonjour est affiché autant de fois.

```
// classe Bonjour2

import java.io.*;
 public class Bonjour2
 {
    public static  void main(String[] args)
    {
    BufferedReader clavier= new BufferedReader(new InputStreamReader(System.in));
     String Saisies;
     int n,fois=0;
     System.out.println("Combien de fois par jour dites-vous Bonjour ?");
       try
         {
          Saisies=clavier.readLine();
          fois= Integer.parseInt(Saisies);
         }
         catch(IOException e)
         {
          System.out.println("Erreur IO");
          System.exit(0);
         }
       for(n=0;n<fois;n++)
       {
        System.out.print("Bonjour numero ");
        System.out.println(n+1);
       }
    }
 }
```

Pour saisir une valeur numérique au clavier, ce n'est pas très simple.

- **System.in** est un objet créé automatiquement à l'exécution d'un programme Java. Il permet d'accéder au clavier mais il est assez élémentaire et on l'utilise systématiquement pour fabriquer un objet plus évolué

- Le package **java.io.*;** contient les classes nécessaires pour créer un objet évolué pour accéder aux données tapées au clavier. La classe **BufferedReader** est ici utilisée pour lire le clavier.

- L'objet **BufferedReader** associé au clavier est appelé ici **clavier**. (Le nom clavier de cet objet choisi ici est arbitraire).

- L'obtention de l'objet **clavier** n'est pas simple :

```
clavier= new BufferedReader(new InputStreamReader(System.in));
```

On passe obligatoirement par un objet intermédiaire de type InputStreamReader.

- La saisie elle-même est obtenue par la méthode **readLine() de l'objet clavier.**

```
Saisies=clavier.readLine();
```

- La saisie est obligatoirement sécurisée par **exception,** avec les blocs **try, catch().**
  Si la saisie est effectuée normalement, les instructions du bloc **try** sont exécutées.
  S'il y a un problème au niveau de la saisie, la JVM déclenche une **IOException**, interceptée dans le bloc **catch()**. C'est alors les instructions du bloc catch qui sont exécutées.
  On n'a pas le choix d'accepter ou de refuser le traitement par exception dans le cas de la saisie au clavier. Java l'impose.

- Les données sont saisies en **String.** Elles doivent être converties en **int** par la méthode **parseInt()** de la classe **Integer**.
  Si vous n'entrez pas une valeur numérique valide au clavier, une exception **java.lang.NumberFormatException** sera levée et par défaut le programme se terminera automatiquement, car nous n'interceptons pas l'exception dans un bloc try catch().

Si tout cela s'est bien passé, une valeur numérique dans la variable **int fois** permet de réaliser des affichages dans **une boucle for**.

Exercice: Réaliser cette application.

Exercice: Sécuriser cette application en interceptant. NumberFormatException dans un bloc try catch()

4. Troisième exemple.

Décomposition du programme en plusieurs méthodes : les programmes précédents sont pauvres vis-à-vis de la POO (Programmation Orientée Objet, La classe Bonjour2 n'a qu'une méthode, la méthode **main()**), **il est maintenant important de faire apparaître plusieurs méthodes (fonctions membres) dans la classe Bonjour.**

```
import java.io.*;
public  class Bonjour3
  {
   public static int Lire()
   {
        BufferedReader clavier= new BufferedReader(new InputStreamReader(System.in));
        String Saisies;
        int fois=0;
        System.out.println("Combien de fois par jour dites-vous Bonjour ?");
        try
        {
           Saisies=clavier.readLine();
           fois= Integer.parseInt(Saisies);
        }
        catch(IOException e)
        {
           System.out.println("Erreur IO");
           System.exit(0);
```

```
        }
      return fois;
   }

public static void afficher(int nfois)
   {
        for(int n=0;n<nfois;n++)
        {
           System.out.print("Bonjour nø ");
           System.out.println(n+1);
        }
   }

   public static  void main(String[] args)
     {
        int nombre;
        nombre=Lire();
        afficher(nombre);
     }
  }
```

L'application Bonjour3 est décomposée en **3 méthodes : main(), Lire() et Afficher(),** toutes les 3 **static**.

Exercice:
Tester le fonctionnement de ce programme.
Essayez d'enlever static d'une des méthodes Lire() ou Afficher(), et interprétez les messages d'erreur.

Réponse: une méthode static ne peut accéder directement qu'aux autres méthodes et attributs déclarés static.
Le seul cas où on peut utiliser un attribut/méthode d'une classe sans avoir à l'instancier est de le/la déclarer **static**.
Exemple

```
public class  A {
          public static void f() { ----}
          ---
}

// Utilisation de f() dans une méthode d'une autre classe :
          A.f() ;
```

Voir la classe **Integer** utilisée paragraphes 2 et 3.

La bonne manière pour programmer en POO est de créer des objets pour accéder à leurs attributs (données membres) et à leurs méthodes (fonctions membres).

5. Quatrième exemple : instancier un objet Bonjour depuis le main().

En restant dans le contexte statique du main(), on ne peut utiliser que des méthodes statiques de la classe. Ce n'est pas la meilleure des façons pour programmer en Java.

On peut instancier des objets sur cette classe depuis le main() en utilisant l'opérateur **new**.
On peut alors accéder aux méthodes/attributs de l'objet en écrivant le nom de l'objet suivi d'un point, lui-même suivi de la méthode/attribut utilisé.

Nous aurons les instructions suivantes dans la méthode main() :

- Déclaration d'un objet de nom b1 de type Bonjour4 :

```
Bonjour4  b1 ;
```

- Création de l'objet (instanciation) avec l'opérateur **new** :

```
b1= new Bonjour4();
```

- Appel des méthodes de la classe avec l'opérateur . (point) :

```
b1.Lire() ;
b1.Afficher() ;
```

```
import java.io.*;
public class Bonjour4
{
public int Lire()
  {
      BufferedReader clavier= new BufferedReader(new InputStreamReader(System.in));
      String Saisies;
      int fois=0;
      System.out.println("Combien de fois par jour dites-vous Bonjour ?");
      try
      {
         Saisies=clavier.readLine();
         fois= Integer.parseInt(Saisies);
      }
      catch(NumberFormatException e)
      {
         System.out.println("Erreur Format");
         System.exit(0);
      }
      catch(IOException e)
      {
         System.out.println("Erreur IO");
         System.exit(0);
      }
    return fois;
  }
 public void afficher(int nfois)
  {
      for(int n=0;n<nfois;n++)
      {
         System.out.print("Bonjour nø ");
         System.out.println(n+1);
      }
```

```
  }
public static  void main(String[] args)
    {
       int nombre;

         Bonjour4  b1 ;
         b1 = new Bonjour4();
         nombre=b1.Lire();
         b1.afficher(nombre);
    }
 }
```

Ecrire et tester l'exemple Bonjour4

6. Constructeur d'une classe

Un constructeur est une méthode qui porte **le même nom que la classe**, peut ou pas avoir d'arguments.
C'est une méthode appelée automatiquement quand un objet est instancié (créé) sur une classe.
Un constructeur est sans type, il ne retourne aucune valeur, ni même void.
Un constructeur sert souvent à faire l'initialisation des attributs de la classe.
Un constructeur par défaut est automatiquement créé par la JVM si aucun constructeur n'est écrit dans la classe. C'est le cas de tous les exemples précédents. Java crée un constructeur par défaut pour créer les objets.

Ajoutons un constructeur dans la classe Bonjour4 :

```
public class Bonjour4
{
 public Bonjour4()
  {
   System.out.print("Je suis le constructeur de Bonjour4 ");
   }
public int Lire()
  {
  ….
} // fin de la classe
```

Dans le cas de l'application Bonjour4, ci-dessus, le constructeur Bonjour4() exécute un code, très simple..

Exercice

α) Tester l'exemple Bonjour4 modifié avec le constructeur. Interpréter les messages affichés.

β) Supprimer le constructeur Bonjour4() en le plaçant en commentaires. Compiler et tester. Comparer avec a).

χ) La variable nombre est déclarée dans le main. Modifier le programme afin de supprimer cette variable nombre et la remplacer par un attribut nombre de la classe Bonjour4, nombre sera initialisé à 0 par le constructeur.

```
public class Bonjour4
{
 int nombre ;              // nombre est ici un attribut de la classe
  ….
} // fin de la classe
```

7. Ecriture de l'application en 2 fichiers

La POO consiste en 1er à définir des classes appelées classes métier pour manipuler un ensemble cohérent de données.
Puis en 2ème on crée l'application qui utilise ces classes métier. C'est obligatoirement une classe de l'application qui contient la méthode main().

Exercice Décomposer le programme en **2 classes** :

- Une classe **Bonjour5.java**, contenant tout le code de la classe Bonjour4 de la page 9 à l'exception de la méthode main (sorte de classe métier).
- **Une classe TestBonjour5**, enregistrée dans un fichier **TestBonjour5.java** qui ne contiendra que la méthode main() de l'exercice précédent :

```
public class  TestBonjour5 {
   public static  void main(String[] args)
     {
          int nombre;
          Bonjour5  b1 ;
          b1 = new Bonjour5();
          nombre=b1.Lire();
          b1.afficher(nombre);
     }
  }
```

Compilation
javac Bonjour5.java
javac TestBonjour5.java

Exécution
java TestBonjour5

8. Qu'est-ce qu'une classe ?

Il n'existe pas de réelle définition d'une classe en programmation objet.
Une classe sert à fabriquer des objets, on peut alors imaginer qu'une classe est un moule.

Un objet est caractérisé par un certain nombre de propriétés.

Par exemple, pour un objet rectangle on aurait les propriétés suivantes :

- Largeur
- Longueur
- Position du centre (X,Y) dans le plan.
- Angle de rotation (si le rectangle n'est pas en position horizontale).

Cet objet rectangle peut être inanimé (il ne bouge plus dans le plan) mais, en informatique, on peut lui faire subir des opérations comme :

- translater
- tourner
- agrandir
- diminuer
- afficher

On définit ainsi une classe Java pour tous nos objets de type Rectangle à créer.

Ecriture simplifiée en Java de la classe **Rectangle**.

```
public class Rectangle {
                    /* Les propriétés = attributs = données membres */
          float Largeur ;
          float Longueur ;
          //Position du centre (X,Y) dans le plan.
          int X ;
          int Y ;
          float Angle ;
                    /* Les opérations = méthodes = fonctions membres */
          void translater(int x, int y)  { /* A coder */ }
          void tourner(float a)  { /* A coder */ }
          void agrandir(float a)  { /* A coder */ }
          void diminuer(float a)  { /* A coder */ }
          void afficher()  { /* A coder */ }
          }
```

Les attributs sont des variables particulières car ils sont déclarés dans la classe mais à l'extérieur de toutes les méthodes. C'est pour cela qu'on dit qu'ils sont les propriétés de la classe.
Les attributs sont accessibles par toutes les méthodes de la classe.

On ne code ici que les méthodes **translater()** et **afficher()**.

```
void translater(int x, int y)  {
        X = X + x ;
        Y = Y + y ;
 }
void afficher()  {
        System.out.println("Ma longueur= "+Longueur+ " et ma largeur= "+Largeur) ;
        System.out.println("Je suis en X = "+X+ " et en Y = "+Y) ;
}
```

Les méthodes afficher() et translater() accèdent aux attributs X et Y.

9. Créer (instancier) les objets

La classe Rectangle est le moule utilisé pour fabriquer des objets de type rectangle.
Ces objets sont des objets informatiques.

Il faut maintenant créer des objets de type Rectangle à partir de la classe précédente.
Cela peut se faire en 2 étapes :

1) Déclarer une variable de type Rectangle : c'est notre objet, il s'appelle ici monrec.

```
Rectangle  monrec;
```

2) Créer l'objet avec l'opérateur **new** :

```
monrec = new Rectangle() ;
```

Où créer les objets ?

La méthode main vue dans les chapitres précédents doit être utilisée pour créer (instancier) des objets de type Rectangle.
On crée pour cela une classe d'application **MainRectangle** qui est ici très simple car elle ne contient que la méthode main().
On suppose travailler dans un même dossier, c'est à dire que les fichiers Rectangle.java et MainRectangle.java sont placés dans un même dossier ainsi que les fichiers .class correspondants..

```
public class MainRectangle {
public static  void main(String[] args)
    {
        Rectangle  monrec;
        monrec = new Rectangle() ;
    }
 }
```

Comment utiliser les méthodes (fonctions membres) de la classe Rectangle ?

Par exempel, le nom de l'objet (monrec) suivi d'un point (.) suivi du nom de la méthode (afficher). On ajoute alors les 3 instructions écrites en gras.

```
public class MainRectangle {
public static  void main(String[] args)
    {
        Rectangle  monrec;
        monrec = new Rectangle() ;
        monrec.afficher() ;
        monrec. translater(10,15) ;
        monrec.afficher() ;
    }
 }
```

Exercice :
Ecrire les classes Rectangle.java et MainRectangle.java dans un même dossier.

Compiler les 2 classes.
Exécuter MainRectangle.
Que valent les attributs de l'objet Rectangle créé?

Réponse : l'objet monrec est bien créé mais tous ses attributs sont mis à 0...

Le constructeur de la classe Rectangle

La JVM crée un constructeur par défaut, l'exemple précédent montre que l'objet monrec est bien créé mais que tous ses attributs sont mis à 0...
Cet exemple montre bien l'utilité d'un constructeur. On pourra alors choisir des dimensions, une place dans le plan et une orientation par défaut.

```
public class Rectangle {
                float Largeur ;
                float Longueur ;
                //Position du centre (X,Y) dans le plan.
                int X ;
                int Y ;
                float Angle ;
  public Rectangle() {
        Largeur = 10 ;
        Longueur = 20 ;
        X= 10 ;
        Y = 5 ;
        Angle = 0 ;
        }
…....
}
```

Exercice :
Modifier la classe Rectangle en ajoutant le constructeur ci-dessus.
Compiler Rectangle.java.
Exécuter MainRectangle. Relever les valeurs affichées. Conclure

Modifier le constructeur afin de lui passer comme arguments la longueur et la largeur.
Tester le fonctionnement.

10. La classe Scanner

La classe Scanner est pratique pour les saisies au clavier.

Exemple simple :

```
import java.util.Scanner;

public class TestScanner  {

 public static void main(String[] args){

   Scanner clavier = new Scanner(System.in);
```

```
// Pour saisir une chaîne de caractères au clavier
   System.out.println("Tapez une chaîne :");
   String s = clavier.nextLine();
    System.out.println(s);

// Pour saisir un entier
    System.out.println("Tapez un entier :");
    int i = clavier.nextInt();
    System.out.println(i);

/* Pour être sûr de saisir un entier : clavier. hasNextInt() retourne true quand un entier est tapé
et false pour tout autre type de données tapé */

   System.out.println("Tapez un entier :");
    while (clavier.hasNextInt() == false) {
                    s = clavier.nextLine();
                   System.out.println("Tapez un entier :");
    }
  int j= clavier.nextInt();
  System.out.println(j);
  }
}
```

Exercices
Etudier et tester le programme ci-dessus.
Modifier la classe Bonjour4 afin d'utiliser la classe Scanner pour effectuer les saisies au clavier.

11. Exercices

11.1 Testeur de poids idéal. Calcul de l'IMC (indice de masse corporelle)

Entrer votre taille en m : exemple 1.80 pour une taille de 1,80 mètre.
Entrer votre poids en kg : exemple 78 pour un poids de 78 kg.

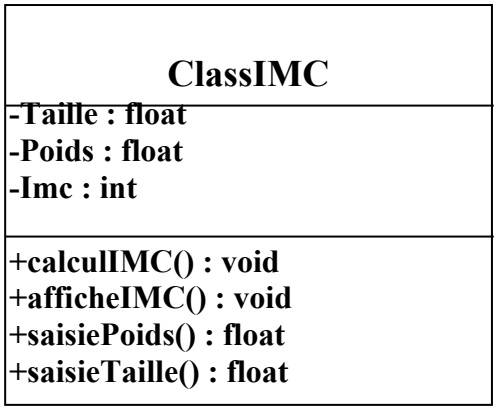
Le programme calcule une valeur appelée **IMC**: poids en kg divisé par la taille au carré, en m :

Soit IMC = 78/ (1.80 * 1.80) puis on arrondit en entier.

L' IMC vaut 24 dans cet exemple.

On compare l' IMC aux valeurs suivantes :

IMC >35 Obèse
30< IMC <=35 Trop gros
27< IMC <=30 Attention



23< IMC <=27 Poids idéal
18< IMC <=23 Mince
15< IMC <=18 Maigre
15< =IMC Anorexique

Exemple d'exécution de l'application :

Entrez votre taille en mètres:
1.80 ←
Entrez votre poids en kilos: ← Valeurs tapées par l'utilisateur
78 ←
votre taille 1.8
votre poids 78.0
Votre IMC = 24
Poids idéal!

Le langage java ressemble au langage C pour les tests : if, else, switch, case

Il faut utilisé la méthode **parseFloat(String s)** de la classe **Float** pour convertir un String en flottant :**Float.parseFloat(String s)** retourne l'argument s convertit en réel.

Taille, Poids et Imc sont les attributs de la classe **classIMC**.

L'attribut Imc sert à ranger le résultat du calcul de l'IMC.

calculIMC(), **afficheIMC()**, **saisiePoids()** et **saisieTaille()** sont les méthodes de la classe.
Le poids et la taille sont saisis au clavier à l'aide des méthodes saisiePoids() et saisieTaille().

11.2 Une classe qui contient des méthodes pour effectuer quelques calculs mathématiques à partir d'une valeur entière

Entier
-valeur :int
+Entier(val :int) **+factorielle() :int** **+afficheValeur() : void** **+somme() : int** **+puissance(y:int) :int**

Diagramme de classes UML de la classe Entier
valeur : Attribut ou donnée membre.

factorielle() : méthode ou fonction membre de la classe.
afficheValeur() : méthode ou fonction membre.
somme() : méthode ou fonction membre de la classe
puissance() : méthode ou fonction membre de la classe.
Entier() est le constructeur de la classe.
L'attribut entier valeur sert à calculer la somme, la puissance et la factorielle.

Voici une description des différentes méthodes de cette classe.

Le constructeur **Entier(int val)**
Affecte l'argument « val » à l'attribut « valeur ».

La méthode **factorielle()**
La méthode **factorielle()** retourne un entier égal à la factorielle de « valeur ».
Rappel : factorielle n = n! = n * (n-1) * (n-2) * ... * 2 * 1
Il y a N nombres à multiplier donc N-1 multiplications à faire. Voir la boucle while.

La méthode **somme()**
La méthode **somme()** calcule somme de « valeur » et retourne le résultat.
Rappel : $\sum N = N + (N-1) + (N-2) + \ldots + 2 + 1$
Il y a N nombres à additionner donc N-1 additions à faire. Voir la boucle while.

La méthode **afficheValeur()**
La méthode **afficheValeur()** se contente d'afficher l'attribut valeur.

La méthode **puissance(int y)**
La méthode **puissance()** calcule « valeur » à la puissance y et retourne le résultat.

Ecrire le programme TestEntier.java pour tester la classe Entier

La fonction main() dans TestEntier.java doit effectuer les principales actions suivantes :
- instancier (créer) un objet de type Entier,
- tester les calculs demandés,
- afficher les résultats,
- et pour finir proposer un menu pour sélectionner l'opération à effectuer.

Ecrire le code de la fonction main(). Tester le fonctionnement.

La boucle while(condition) permet de répéter l'exécution d'un bloc d'instructions tant que la condition est vraie.

Syntaxe :
```
while (condition) {
        /* Les instructions à exécuter : le bloc d'instructions */
        }
```

Exemple :
```
        i=0 ;
        while (i < N) {
                System.out.println(" Parcours numéro "+i);
                i=i+1;
                }
```

Si N vaut 3, l'affichage donne :
Parcours numéro 0
Parcours numéro 1
Parcours numéro 2