Bases de JAVA

- La syntaxe de JAVA est volontairement très proche de celle de C/C++
- En JAVA, on manipule deux types d'entités :
- Les **variables** qui sont d'un *type primitif* défini
  - Elles sont créées par *déclaration* (ex: "`int i ;`" déclare une variable de nom `i` et de type `int`)
- On peut leur *affecter* une valeur de leur type (ex: "i=0;") et/ou les utiliser dans une expression (ex: "i = 2*i + 1 ;")
- Les **objets** via des *références*
- Les objets sont des *instances de classes*
- Ce sera étudié en détail au chapitre suivant !

2. Les bases : généralités sur JAVA

Organisation Générale

- L'unité de base d'un programme JAVA est la *classe* :

```
class UneClasse
 { définition des attributs
   définition des méthodes
 }
```

- Les classes sont décrites dans des *fichiers* :
  - Un fichier peu contenir plusieurs classes, mais une seule avec l'attribut `public` (à placer devant le mot clé `class`) qui sera visible depuis tous les autres fichiers et qui donnera son nom au fichier
  - Un fichier peut débuter par une déclaration de paquetage (ex : `package monPaquetage;`) et/ou des importations de classes venant d'autres paquetages (`import MaClasse;`)

Les types primitifs

On retrouve en JAVA la plupart des types du C/C++, avec quelques changements et la disparition des non signés (`unsigned`). Les types ne sont pas des objets, des *wrappers* font la correspondance.

Type	Wrapper	Taille (octets)	Valeur minimale	Valeur maximale
byte	Byte	1	-128 Byte.MIN_VALUE	127 Byte.MAX_VALUE
short	Short	2	-32768 Short.MIN_VALUE	32767 Short.MAX_VALUE
int	Integer	4	-2147483648 Integer.MIN_VALUE	2147483647 Integer.MAX_VALUE
long	Long	8	-9223372036854775808 Long.MIN_VALUE	9223372036854775807 Long.MAX_VALUE
float	Float	4	-1.40239846E-45 Float.MIN_VALUE	3.40282347E38 Float.MAX_VALUE
double	Double	8	4.9406564584124654E-324 Double.MIN_VALUE	1.797693134862316E308 Double.MAX_VALUE
char	Character	2	! Character.MIN_VALUE	? Character.MAX_VALUE
boolean	Boolean	1 Pas d'ordre ici	false Boolean.FALSE	true Boolean.TRUE

Conventions de codage

Non obligatoires mais à respecter

- Le nom d'une classe commence par une majuscule (exemple : `UneClasse`)
- Le nom d'un méthode commence par une minuscule (exemple : `uneMethode`)
- Chaque mot supplémentaire d'un nom composé d'une classe ou d'une méthode commence par une majuscule (classe : `UnPetitExempleDeClasse`, méthode : `unPetitExempleDeMethode`)
- Les constantes sont en majuscules avec le caractère *souligné* (*underscore*) "_" comme séparateur (exemple : `MIN_VALUE`)

Le type String

JAVA possède un vrai type chaîne de caractères

- Déclaration et initialisation similaires aux types primitifs

```
String chaine="une chaine" ;
String chaine2=new String("une chaine") ; // Même chose
String chaine3=chaine ;  // Utilisation d'autres chaînes
```

- Les valeurs des chaînes ne sont pas modifiables

```
String chaine="une chaine" ;
chaine="salut" ; // En réalité il y a ici création d'un nouvel objet
```

- Nombreuses possibilités de manipulation

```
c=2+chaine+"s" ; // Donne 2saluts (+: opérateur de concaténation)
int n=c.lenth() ; // Donne 7 (méthode longueur)
char x=c.charAt(2) ; // Donne a (caractère situé à la position 2)
boolean b=c.equals("autre") ; // Donne false (comparaison)
int p=c.compareTo("autre") ; // Equivaut à strcmp() du C/C++
```

etc.

Les références

Les objets se manipulent en JAVA via des *références*

- Les références sont en réalité des *pointeurs* (au sens des langages C/C++) bien cachés
- On *déclare* une référence comme pour les types primitifs

```
Integer entier ;
```

- La déclaration ne réserve pas la place mémoire pour l'objet, elle sera allouée sur demande explicite par l'opérateur ***new***

```
entier = new Integer() ;
```

- La valeur null désigne une référence non allouée
- La libération de la mémoire est gérée par la JVM

Opérateurs (entre autres...)

Les opérateurs sont ceux de C/C++

- Arithmétiques :
  +,-,*,/,%
- Incrémentation :
  ++,-- ,+=,-=,/=,*=, %=
- Relationnels :
  == (egal),!= (différent),>,<,>=,<=
- Booléens :
  && (ET), || (OU), ! (NON)
- Binaires :
  & (ET), | (OU), ^ (XOU), ~ (complément), >>, <<

Les tableaux

Les tableaux se manipulent en JAVA comme des objets

- Les tableaux sont déclarés comme des références

```
int t [] ; // t sera une référence à un tableau d'entiers
int [] t ; // même chose
int [] t1, t2 ; // t1 et t2 sont des tableaux d'entiers
int t1[], n, t2[] ; // même chose pour t1 et t2, et n est entier
Integer ti [] ;  // ti est un tableau d'objets de type Integer
```

- On crée un tableau comme on crée un objet : avec ***new***

```
t = new int[5] ; // t fait référence à un tableau de 5 entiers
ti = new Integer[3] ; // ti: tableau de 3 objets Integer
ti[0] = new Integer() ; // on est pas dispensé d'allouer les objets
```

- On peut initialiser les tableaux à la déclaration

```
int t[] = {1,n,4} ; // t[0] vaut 1, t[1] vaut n... plus besoin de new
```

- Les tableaux sont indexés à partir de zéro comme C/C++

Structures conditionnelle if (2/2)

```
// Exemple de code et d'execution
public class EssaiIf
{ public static void main (String[] args)
  { int a = 2, b = 3, max ;

    if (a < b)
      max = b ;
    else
    { if (a == b) // Ce "if" n'a pas de "else"
      { System.out.println("Les nombres sont egaux !") ;
        System.exit(0) ; // Pour sortir du programme
      }
      max = a ;
    }
    System.out.println("maximum = "+max) ;
  }
}
---
maximum = 3
```


Structures de contrôle

Les structures de contrôle sont celles de C/C++

- On appelle *suite* une suite d'une ou plusieurs instructions consécutives (ex : `i = 0 ; j = i ;`)
- On appelle *bloc* une instruction seule (ex : `i = 0 ;`) ou une suite d'instructions placées entre { et } (ex : {`i = 0 ; j = i ;` })
- On retrouve les structures conditionnelles :
  - **`if ... else ...`**
  - **`switch`**
- On retrouve les structures de boucles :
  - **`for`**
  - **`while`**
  - **`do...while`**

Structure conditionnelle switch (1/2)

L'instruction **`switch`** cherche le résultat d'une *expression* (de type **`char`** ou **`int`**) dans une liste de cas et exécute la suite correspondante puis toutes les suites suivantes :

```
switch (expression)
{case cas1:suite1 case cas2:suite2 ... case cas_n:suite_n}
```

Optionnellement le dernier cas peut être le mot clé **`default`** si aucun cas ne correspond :

```
switch (expression)
{case cas1:suite1 case cas2:suite2 ... case cas_n:suite_n
default:suite}
```

Pour sortir du **`switch`** à la fin d'une suite d'instructions, utiliser **`break;`** en dernière instruction de cette suite

Structure conditionnelle if (1/2)

L'instruction **`if`** teste une condition booléenne, si le résultat est vrai, le bloc d'instructions suivant la condition est exécuté, sinon il ne l'est pas :

```
if (condition) bloc1
```

Optionnellement, le premier bloc sera suivit du mot clé `else` et d'un second bloc exécuté seulement si la condition est fausse :

```
if (condition) bloc1 else bloc2
```

Structure de répétition while

L'instruction **while** évalue une *condition* puis répète les instructions du *bloc* tant que cette condition est vraie

while (condition) bloc

// Exemple de code et d'execution

```
public class EssaiWhile
{ public static void main (String[] args)
  { int i = 0 ;
    while (i<3)
    { System.out.println("i vaut "+i) ;
      i++ ;
    }
  }
}
---
i vaut 0
i vaut 1
i vaut 2
```


Structure conditionnelle switch (2/2)

// Exemple de code et d'execution

```
public class EssaiSwitch
{ public static void main (String[] args)
  { int a = 1 ;

    switch (a)
    { case 0  : System.out.println("NUL") ; break ;
      case 1  : System.out.println("MOYEN") ;
      case 2  : System.out.println("GRAND") ; break ;
      default : System.out.println("Inconnu") ;
    }
    System.out.println("Fin") ;
  }
}
---
MOYEN
GRAND
Fin
```


Structure de répétition do...while

L'instruction **do...while** exécute une *suite* puis évalue une *condition* et répète *bloc* tant que *condition* est vraie

do suite **while** (condition) ;

// Exemple de code et d'execution

```
public class EssaiDoWhile
{ public static void main (String[] args)
  { int i = 0 ;
    do
    { System.out.println("i vaut "+i) ;
      i++ ;
    }
    while (i<3) ;
  }
}
---
i vaut 0
i vaut 1
i vaut 2
```


Structure de répétition for

L'instruction **for** exécute une *instruction1* puis répète les instructions du *bloc* suivant l'instruction **for** tant que sa *condition* est vraie, à la fin de chaque répétition elle exécute son *instruction2* :

for (instruction1;condition;instruction2) bloc

// Exemple de code et d'execution

```
public class EssaiFor
{ public static void main (String[] args)
  { for (int i=0;i<3;i++)
    System.out.println("i vaut "+i) ;
  }
}
---
i vaut 0
i vaut 1
i vaut 2
```