# Récapitulatif Java - langage

- **Déclaration des variables**:
  - toute variable doit être déclarée avant d'être utilisée
  - la visibilté de la variable est le bloc où elle est est déclarée ainsi que tous les blocs imbriqués
  - une variable peut-être déclarée n'importe où dans le programme (il n'y a pas de notion de bloc de déclaration)
  - syntaxe: le type précède le nom de la variable.
- Une variable peut être initialisée lors de sa

  déclaration. Ex: int i = 2
- En Java, il y a 8 types primitifs:

  4 pour le entiers: **byte**, **short**, **int**, **long**;
  2 pour les réels: **float**, **double**;
  1 type booléan: **boolean**;
  1 type caractère: **char**.
- Tous les autres types sont des classes.
- Déclaration des tableaux

  un tableau est un objet. Il faut donc premièrement le déclarer puis l'instancier avec l'opérateur d'allocation **new**. C'est au moment de la déclaration que le type et l'arité sont spécifiés mais c'est au moment de l'allocation que la taille est fixée.

  Ex: int ti[][] = new int[10][20] déclare un tableau ti de 10 lignes et 20 colonnes d'entiers.
- **Les opérateurs**

- Les opérateurs d'affectation: modifie la valeur d'une valeur. Ce sont =, +=, -=, *= et /=
- Autoincrément et autodécrément:
  ++ additionne 1, -- soustrait 1 ; ex : i++ ; j--
- Opérateurs arithmétiques binaires:
  +, -, *, / et % (modulo, i.e. reste de la division entière). Lorsqu'utilisé sur deux opérandes de type entier, / à la signification de la division entière.
- Opérateurs d'égalité:

  == et != sont utilisés pour tester l'égalité (resp. l'inégalité) de deux valeurs. Le résulat est de type booléen.
- Opérateurs relationnels:

  <, <=, >, >=. Le résulat est de type booléen.
- Opérateurs logiques:

  **!** (négation), **&** (et), **&&** (et évalué), **|** (ou), **||** (ou évalué), **^** (ou exclusif)
- Opérateurs logiques **évalués**:

  si le résultat de l'expression logique peut-être déterminé après l'évaluation du premier terme de l'expression, les autres termes ne sont pas évalués; sinon, on continue par l'évaluation du second terme et ainsi de suite.
- **Instructions**
- Séparateur d'instruction :

  le **;** sépare les instructions (tout comme en Oberon)

- Bloc:

  séquence d'instructions délimitée par des accolades { }

- Les instructions de contrôle **for**, **if**, **do** et **switch case** portent sur une seule instruction. S'il faut contrôler plus d'une instruction, utiliser un bloc.

- instruction **for**:

  itération d'instructions (ressemble à **for** en Oberon). La condition d'itération est spécifiée par trois expressions: la valeur de départ, la condition d'itération et l'expression d'incrément. Par ex: for (i=0; i < 10; i++) ... La condition est délimitée par des **( )** et les expressions séparées par des **;**

- instruction **continue**:

  continue passe directement à l'itération suivante dans une boucle for.

- instruction **if**...**else**:

  exécution conditionnelle d'instructions (ressemble à **if** en Oberon mais il n'y a ni "then" ni "elsif" ). La condition s'écrit obligatoirement entre **( )**.

- instruction **while**:

  itération d'une instruction tant que la condition est vraie . La condition est testée avant l'exécution de l'instruction (similaire à "while...do" d'Oberon). La condition s'écrit entre **( )**.

- Instruction **do**...**while**:

  itération d'une instruction tant que la condition est vraie. La condition est testée après l'exécution de

l'instruction (similaire à "repeat...until" d'Oberon). La condition s'écrit entre **( )**.

- Instruction **switch** ... **case**:

  instruction "d'aiguillage" en fonction de valeurs entières (ressemble à case ... of d'Oberon). Attention l'exécution continue après l'embranchement. Si le comportement du case...of d'Oberon est désiré, utiliser **break** à la fin de chaque embranchement.

- Instruction **break**:

  interrompt l'exécution d'une instruction de contrôle. L'exécution continue avec l'instruction qui suit directement le bloc définit par l'instruction de contrôle.

- **Méthodes**

  c'est l'équivalent Java des procédures (et fonctions).

- Déclaration de la méthode:

  le type du résultat retourné par la méthode (fonction) précède directement le nom de la méthode. Si la méthode ne retourne pas de résultat (procédure), le nom est précédé par **void**.

- Déclaration des paramètres de la méthode:

  les paramètres suivent le nom de la méthode et sont délimité par des **( )** (idem Oberon).

- Le passage des paramètres se fait toujours par valeur
- Le corps de la méthode suit immédiatement l'entête et est délimité par des accolades **{ }** (bloc)
- L'instruction **return**: retourne la valeur de la méthode

et termine l'exécution de la méthode (idem Oberon)

- Méthode **static**: méthode de classe (et non d'instance). C'est l'équivalent d'une procedure ou d'une fonction Oberon.
- Méthode qui s'appelle **main**: c'est la méthode par laquelle va commencer l'exécution du programme. Correspond à l'exécution classique d'un programme.
- Déclaration d'un programme (méthode main): un programme est déclaré dans une **classe**.

## D**éveloppement de programmes avec BlueJ**

- Cycle de développement

  - lancer BlueJ
  - ouvrir ou créer un projet
  - New Class (créer une nouvelle classe)
  - BlueJ génère une variable d'instance et un constructeur et une méthode par défaut
  - éditer le code source de la classe
  - si des classes de l'API sont utilisées : **import** … avant la déclaration **class**
  - Compile (compiler la classe)
  - Clic droit sur la classe
    lancer la méthode **main**
    **ou**
    créer une instance d'objet avec new <nom constructeur>
    clic droit sur l'objet
    lancer une méthode d'instance

  - corriger la classe au besoin
  - compiler
  - lancer la méthode main etc.