Java
Structure du langage

Objectifs

- Connaître les bases de la syntaxe en Java
- Savoir exécuter du code dans la méthode principale (main)

Les commentaires

- // commentaire jusqu'à la fin de la ligne n
  - `// ceci va jusqu'à la fin de la ligne n int i = 1;`
  - `// celui ci aussi`
- n /* … */ blocs de commentaire n
  - ```
    /* commentaire occupant
       deux lignes ou plus... */
    ```
- /** … */ commentaires de documentation (javadoc)
    ```

Les séparateurs

- Une instruction se termine par un point virgule ‘;’
- Elle peut tenir sur plus d’une ligne
  - `total=ix+iy+iz;   //une instruction`
  - ```
    total=ix+
          iy+iz;  //2 lignes, 1 instruction
    ```
    ```

Les séparateurs

- Un bloc regroupe un ensemble d'instruction, il est délimité par des accolades '{ }'

```
{ // un bloc
   ix = iy+1;    // une instruction
   iz = 25;      // la deuxième du bloc
}
```

- Ils peuvent s'emboîter les uns dans les autres

```
{
        int ix = iy+1;
        while (ik<MIN) {
              ik = ik*ix;
        }
}
```

Les identificateurs, les noms de variables

- Il commence par une lettre, un trait de soulignement ‘_’, ou un ‘$’
- Les caractères suivants peuvent comporter des chiffres
- Ils respectent la casse : ‘M’ 1 ‘m’
- Pas de limitation de longueur
- Ne doivent pas être des mots clés du langage
  - ~~2var_5~~
  - identificateur
  - _var_sys fenêtre
  - MaClasse
  - $picsou
  - M5TLEA3245
  - MaClasseQuiDeriveDeMonAutreClasse

Les identificateurs

- Un identificateur de variables sert à manipuler deux sortes de choses
  - Des variables représentant un élément d'un type primitif (cf la suite...)
  - Des variables représentant des références (adresses) à des instances, des objets (cours sur les objets et les classes)

Les types primitifs

- Les entiers
  - byte : 8 bits, 1octet (-128 à 127)
  - short : 16 bits, 2octets (-215 à 215-1)
  - int : 32 bits, 4octets (-231, 231-1)
  - long : 64 bits, 8octets (-263,263-1)
- Peuvent être exprimés en:
  - Décimal, base 10
  - Hexadécimal, base 16
  - Octal, base 8

```
byte count;
short v=32767;
int i=73;
int j=0xBEBE; //48830
long var=077; //63

//Dans une expression
98L -> décimal de type long
077L -> 63 sur un long
0xBEBEL -> 48830 sur un long
```

- Tous les entiers sont par défaut considérés comme des int, pour les voir comme des long il faut suffixer la valeur d'un L ou l

Les types primitifs

- Les réels
  - float : 32 bits, simple précision
    (1.40239846 e-45 à 3.40282347 e38)
  - double : 64 bits, double précision
    (4.94065645841246544 e-324 à 1.79769313486231570 e308)
- Typage très strict, double par défaut, sinon suffixée par F
- `static final float PI=3.141594635 //erreur`
- `static final float PI=(float)3.141594635 //OK`
- `static final float PI=3.141594635F //OK`
- static final : définition d'une constante

Les types primitifs

- La logique, les booléen
  - boolean
  - Deux valeur true ou false
    - boolean isOK = true;
    - boolean isnt = false;
- Les caractères
  - char, codés sur 16 bits, Unicode (\u suivi du code hexadécimal à 4 chiffres du caractère)
  - Entourés de « ' »

```
char c= 'a';            //le caractère a
char tab= '\t'          //une tabulation
char carLu= '\u0240' //caractère unicode
```

Résumé : les types primitifs

- byte, short, int, long
- float, double
- boolean
- char
- Il n’existe pas de type primitif chaîne de caractère en java, la ‘string’ est représentée par un objet de la classe **String**

Variables de type primitif et références

- Distinction entre types primitifs et les autres
  - Types primitifs : valeur stockée directement dans la mémoire
    - `int entier;`
    - `entier = 5;`
    - `entier -> 5`
  - Objets : seule la référence est stockée
    - `Integer entier;`
    - `entier = new Integer(5);`
    - `entier -> référence -> 5`

Initialisation

- En java une donnée doit être obligatoirement initialisée avant d'être utilisée (variables locales), sinon il y a erreur de compilation
- Initialisation automatique pour les variables de classes

Type	Valeur par défaut
`byte, int, short`	`0`
`long`	`0L`
`float`	`0.0f`
`double`	`0.0d`
`char`	`'\u0000'(Null en Unicode)`
`boolean`	`false`

- Si une variable est d'un autre type qu'un type primitif (une classe) on l'initialise à null (aucun objet, la variable ne référence rien).

Typage strict

- Attention le typage est très strict en Java.
- Le compilateur refuse de passer d'un type à un autre s'il y a un risque de perte d'information
- C'est pour cette raison qu'il considère par défaut qu'une constante réelle est un double

```
int i=258; //Ok
long l=i;  //Ok car codé sur plus de bits
byte b=i;  //error:explicit cast need to convert int to byte
byte b2=200; // idem
float f=253.62 // error : possible loss of precision, found
                    double, required float...
```

Forçage de type, le *cast*

- Il peut être nécessaire de forcer le programme à considérer une expression comme étant d'un type différent de son type réel ou déclaré
- On effectue alors le cast de l'expression
- On force le type : `(typeforcé) expression`

```
int a,b;
double division = (double)a / (double)b; // pour forcer la division réelle
```

Forçage de type, le *cast*

- Attention : le cast peut être source de perte de précision voire de données
  - Perte de précision : C'est le cas quand on passe d'un long à un float (on perd des chiffres significatifs, mais on conserve l'ordre de grandeur, troncation)
  - Perte de données : Lorsqu'on passe par exemple d'un int (32) à un byte (8)

```
short s = 24;
char c = s; // Caractère dont le code est 24
int i = 130;
byte b=(byte)i; // Ok mais b=-126
```

Les types énumérés

- Il est possible de construire de nouveaux types en énumérant leur valeur
- Déclaration :
  - `public enum Day{SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY}`
- Utilisation :
  - ```
    ...
    Day day;
    ...
    switch (day){
      case MONDAY: System.out.println("Mondays are bad."); break;
      case FRIDAY: System.out.println("Fridays are better."); break;
      case SATURDAY:
      case SUNDAY: System.out.println("Weekends are best."); break;
      default: System.out.println("Midweek days are so-so."); break;
    }
    ```
    ```

Les opérateurs

- Arithmétiques
  - + - *
  - /{division entière si les opérandes sont entiers}
  - % {modulo}
  - ++ --
- Relationnels
  - ==, !=, >, <, >=, <= (résultat booléen)
- Logiques
  - && || l'opérande 2 est évalué si besoin
  - & | ! ^(ou excl.) les 2 opérandes sont évalués

Les opérateurs

- Opérateurs sur les bits
  - Décalage de bit : >>, <<, >>>
  - Opération bit à bit : &|^ ~(complément à 1)
- Référence : .
- Tableaux : []
- Groupement : () {} ;
- Affectation
  - =
  - += -= *= /= %=
  - ^= ~= |= &= << = >> = >>>= Ex : `Integer.toBinaryString(0b00010001 << 2 )` -> 1000100
- Autre
  - ,
  - ? : (raccourci pour if-else)
  - new
  - instanceof

Les opérateurs

- Trois types d’opérateurs
  - Unaire : !a
  - Binaire : a && b
  - Ternaire : (a > b) ? a : b
- Les expressions sont évaluées de la gauche vers la droite
- N’hésitez pas à mettre des parenthèses pour « forcer » le calcul en fonction des priorités

Les expressions

- Le type d'une expression arithmétique dépend des opérateurs et des types des éléments de l'expression
  - Si les éléments ont tous le même type, pas de problème...
  - Sinon, le type final correspond au type le plus précis (le plus d'octets)

```
float * double = double
```

Les structures de contrôle

If-else, switch

While, do..while, for

Break, continue

return

Conditionnelles : if … else

- Si condition vrai alors instructions1,
  Sinon instructions2

```
if(condition) {
     instructions1;
} else {
     instructions2;
}
```

```
(condition) ? instructions1 : instructions2
```

```
if (i<0) i++;
if (end) {// end doit valoir true
        fin = true;
        i=0;
} else i--;
```

Conditionnelles : switch

- L'expression doit retourner un type énumérable : int, byte, short, boolean, char
  - Si break, on invalide les tests suivants
  - Sinon on continue sur le case suivant

```
switch (expression){
     case exp1 :
          instructions1;
          [break;]
     case exp2 :
          instructions2;
          [break;]
      ...
     default : instructions;
```

Conditionnelles : switch

```
switch (month) {
	case 1: System.out.println("January"); break;
	case 2: System.out.println("February"); break;
	case 3: System.out.println("March"); break;
	case 4: System.out.println("April"); break;
	case 5: System.out.println("May"); break;
	case 6: System.out.println("June"); break;
	case 7: System.out.println("July"); break;
	case 8: System.out.println("August"); break;
	case 9: System.out.println("September"); break;
	case 10: System.out.println("October"); break;
	case 11: System.out.println("November"); break;
	case 12: System.out.println("December"); break;
	default: System.out.println("That's not a valid month!");
break;
}
```

Itératives : for

- Boucle à nombre d'itération connu
- break, sortie de boucle
  - L'exécution reprend après le bloc for
- continue, force l'exécution à l'itération suivante

```
for (init; test; incrément){
   instructions ;
}
```

  - init : initialise la variable compteur
  - Condition de poursuite
  - Evolution du compteur

Itératives : for

- Exemples

```
for (int i=0 ;i<10;i++){
      ...
      if (i==0) continue;//on saute à l'itération i=1
                         //sans exécuter les instructions
                         //suivantes
      if (bonneValI(i)) break; // on sort de l'itération
}
...
int i=0;
for (; i<100; i++){...//Initialisation hors boucle}
...
for ( ; ; ){...//boucle infinie}
```

Itératives : for

- Utilisation avec un ensemble structuré d'éléments

```
for (typeElement element : nomCollection){
     //traitement effectué sur element
}


class EnhancedForDemo{
     public static void main(String[] args){
          int[] numbers = {1,2,3,4,5,6,7,8,9,10};
     for (int item : numbers){
               System.out.println("Count is: " + item);
          }
     }
}
```

Itératives : while

- Boucle à nombre d'itération inconnu

```
while (expression){
      instructions;
}
```

- expression : Condition qui doit être vérifiée pour que les instructions de la boucle soient exécutées

```
int i = 0;
while (i < C.length){
      C[i] = i*i;
      i++;
}
```

- Les instructions de la boucle ne sont exécutées que si la condition est vérifiée au moins une fois

Itératives : do … while

- Boucle à nombre d'itération inconnu, exécutée au moins une fois

```
do{
      instructions;
}while (expression);
```

- expression : Condition qui doit être vérifiée pour que les instructions de la boucle soient ré-exécutées

```
int i=0;
do{
      C[i]=i*i;
      i++;
} while (i<C.length);
```

- Les instructions de la boucle sont exécutées au moins une fois

Exemple: lecture d'une phrase, caractère par caractère

```
public static String lirePhraseClavier()throwsIOException{
      String ligne = " ";
      char c;
      do{
            c = (char)System.in.read(); //lecture d'un
                                        //caractère au clavier
            ligne += c; //reconstruction de la chaîne
      } while (c != '\n' && c != '\r');
      return ligne.substring(0,ligne.length()-1);
}
```

Itérative : les boucles infinies

- for
  - for(; ;){...}
- while
  - while (true) {...}
- do...while :
  - do {...} while (true)

Break et continue

- Les instructions continue et break, sont utilisables dans tous les types de boucles : for, while, do...while
  - continue : passe à l'itération suivante (incrémentation du compteur) sans exécuter les instruction d'après dans la boucle
  - break : sort de la boucle et exécute la première instruction qui suit celle-ci

Les blocs labellisés

- Indique la référence de la boucle que l'on veut stopper ou continuer

```
un : while (cond1){
     deux : for (...){
          trois : while (...){
               if (...)continue un;
               if (...) break deux;
               continue;
```

- L'étiquette d'un « continue » doit se trouver sur une boucle englobante pour indiquer que l'exécution doit reprendre avec cette boucle et non avec celle qui est directement englobante comme c'est le cas par défaut.
- C'est la même chose pour le break

La portée des variables

- Variables locales
  - Définies par block { }
  - Doivent être initialisées explicitement

```
{
      int ix=0; int iz=0;
      // portée de la déclaration...
      { // ici aussi
      }
}
// fin de la portée pour ix et iz
for(int i=0;i>t;i++){
      //portée pour i
}
```

Les conventions

- Conventions de nommage -> CamelCase
  - classe : MaClasse, LaTienne, Rectangle,...
  - variables et méthodes : nombreClients, calculer, calculExact,dessiner,...
  - constante : TAILLE_MAX,MAX_INT,PI,JAUNE
  - package : fr.univ-corse.coursjava.util
- Organisation de fichier
  - Une seule classe public par fichier
  - Une seule méthode main par projet
- Attention : les majuscules sont importantes en Java !

Les méthodes (ou fonctions)

- Syntaxe :
  - `public static type nomMethode(arguments){ … };`
- **public / private** : portée de la méthode : disponible localement ou accessible depuis d'autres classes (voir chapitre suivant)
- **static** : indique que la méthode appartient à lui-même et non à une instance de ce type -> méthode locale et non pas méthode de classe
- **type** : type de retour de la méthode : type primitif (int, float) ou classe (String, MaClasse)
- **arguments** : séparés par des virgules s'il y en a plusieurs