Algorithme

Type :
- booléen
- caractère
- chaîne
- entier
- réel

Déclaration :
var nom : type
const nom ← val : type

var i : *entier*
const pi←3.14 : *rééI*

Affectation :
nom ← expr

i ← 10
i ← i + 10

Lecture :
i ← lire()

Ecrire :
ecrire(expression)

Conditionnelle :
si (expression booléenne) **alors**
    instruction
sinon
    instruction
finsi

rem : la séquence est un instruction et la partie sinon est optionnelle

Boucle tant que :
Tant que (expression booléenne) **faire**
    Instruction
Fin tant que

Boucle faire/tant que :
Faire
    Instruction
Tant que (expression booléenne)

Java

Type :
- boolean
- char
- String
- int
- double

Déclaration :
Type nom
final type nom = valeur

int i ;
final double pi = 3.14 ;

Affectation :
nom = expr

i = 10;
i = i + 10;

Lecture :
Scanner sc = new Scanner(System.in) ;
i = sc.nextInt();

Ecrire :
System.out.println(expression)

Conditionnelle :
```
if   (expression booléenne)
{
    instruction
}
else
{
    instruction
}
```

Boucle tant que :
```
while (expression booléenne)
{
    Instruction
}
```
Boucle faire/tant que :
```
do
{
    Instruction
}
while (expression booléenne)
```

Boucle pour :

```
pour i de deb à fin pas p faire
        instruction
fin pour
```

Opérateurs de comparaison :

>, <, ≥, ≤, =, ≠

Opérateurs booléen :

- ou
- et
- non

Structure d'un programme principal :

```
Algo nomAlgo
 déclarations
Début
 instructions
Fin
```

Déclaration de sous-programme et appel :

```
Fonction nomFonc(param1 : type1 [,
par am2 : t ype2] ) : TypeRetour
 …
 Instructions
 …
 Renvoie (resu l t a t)
 // si nécessaire (TypeRetour != vide)
Fin Fonction

Algo Appel
 var r : TypeRetour
// t ypeRetour ne peu t ici être vide
Début
 r ← nomFonc(val 1 , val2)
 // si la fonction ne retourne rien ,
 //l ' affectation n 'existe pas
Fin
```

Boucle pour :

```
for (i=deb ; i <= fin ; i = i + pas)
{
        instruction
}
```

Rem : deb <= fin sinon il faut les inverser

Opérateurs de comparaison :

>, <, >=, <=, ==, !=

Opérateurs booléen :

- ||
- &&
- !

Structure d'un programme principal :

```
class nomAlgo
{
 public static void main(String [ ] args)
  {
   déclarations
   instructions
  }
}
```

Déclaration de sous-programme et appel :

```
class Appelee
{
 static typeRetour nomFonc(type1 par
am1[, type2 par am2])
  {
   …
   instructions
   …
   return (resultat) ;
  }
}

class Appelante
{
 public static void main(String [ ] args)
  {
   TypeRetour r ;
   r = Appelee.nomFonc(val1 , val2);
  }
}
```