# Initiation à la programmation

Module M1011 - DUT Informatique

# **Notion de répétition**

yann.secq@univ-lille1.fr

Abdelghani ATAMENIA, Géry CASIER, Iovka BONEVA, Antoine NONGAILLARD

*Promotion 2016*

# Structure de contrôle

- Base: la séquence d'instruction
- Nécessité d'influencer le choix des instructions à exécuter
- Les structures de contrôle (du flux):
  - Alternative (choix) et boucle (répétition)

# Structures de contrôle

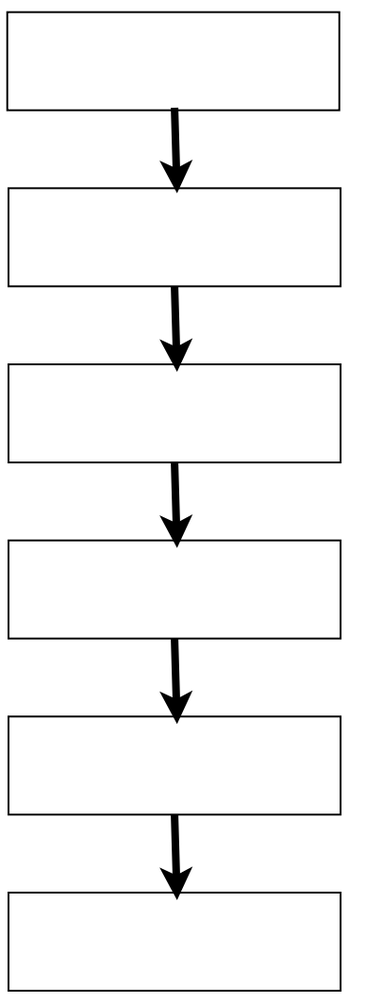
- Les alternatives permettent des branchements
- Les boucles permettent des répétitions
- Boucle: répéter un bloc d'instructions un certain nombre de fois

Affectation + Alternative + Boucle + Fonction

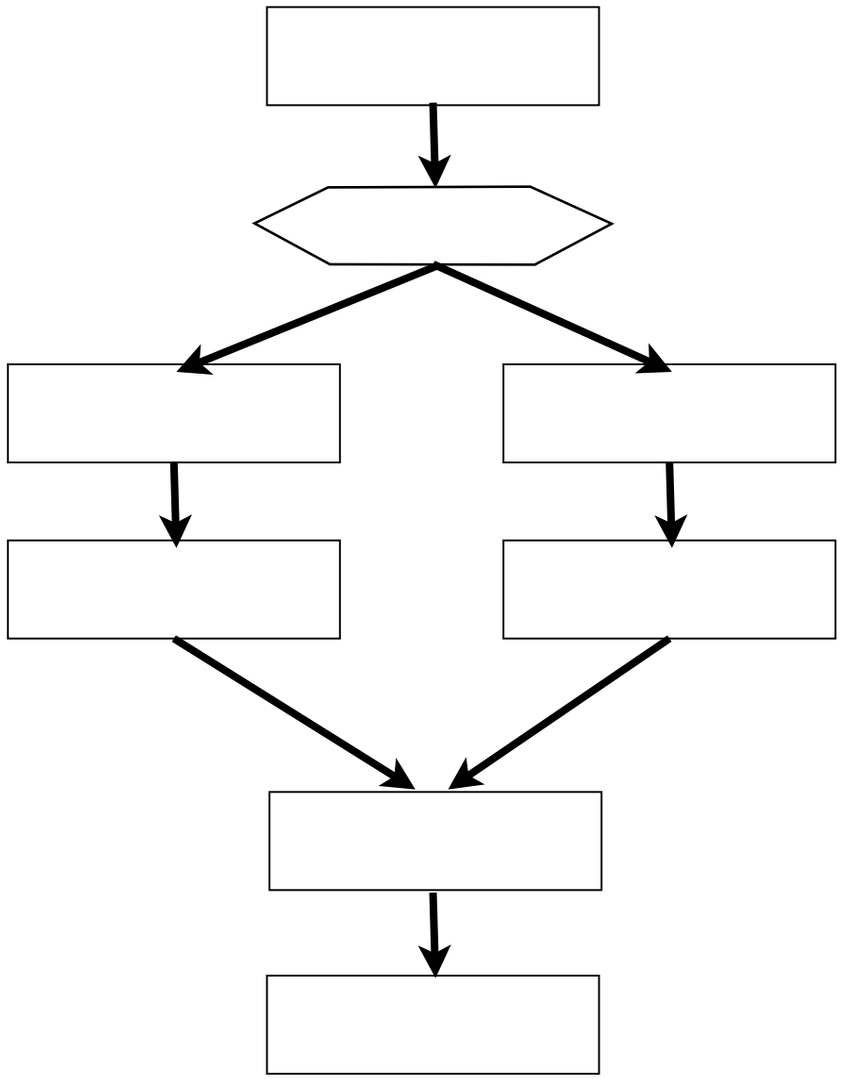
=

Bases de l'algorithmique

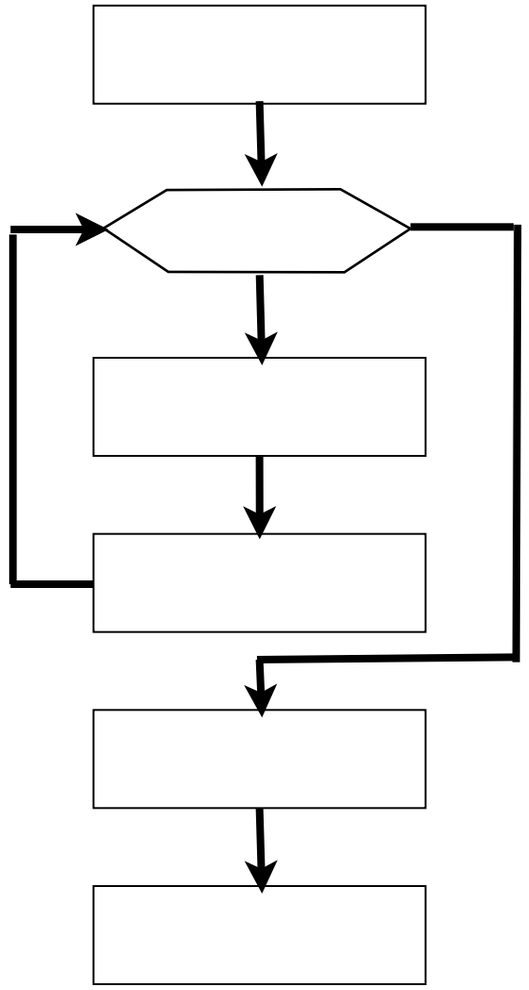
# Structures de contrôle



Séquence



Alternative



Boucle

# Trois types de boucles

```
do {
   <bloc d'instructions>
} while (<condition>) ;
```

```
while (<condition>) {
   <bloc d'instructions>
}
```

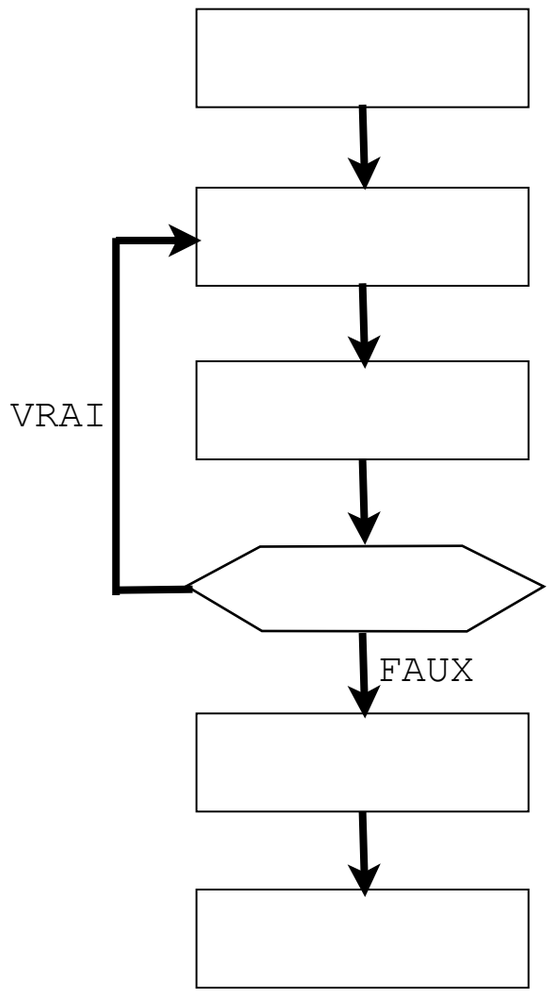
Nb de tours inconnus a priori (détection d'évènement)

```
for (<init cpt> ; <condition> ; <màj cpt>) {
   <bloc d'instructions>
}
```

Nb de tours connus a priori (boucle à compteur)

# Répéter tant que

- Répète un bloc d'instructions tant qu'une condition est vérifiée (ie. `VRAI/true`)
- A utiliser lorsque l'on ne connaît pas a priori le nombre de tours de boucle
- ATTENTION: la condition est évaluée **en fin** de boucle




```
do {
   <bloc d'instructions>
} while (<condition>) ;
```

# Dynamique du `Répéter`

```
do {
   <bloc d'instructions>
} while (<condition>) ;
```

1. Le bloc d'instruction est évalué
2. Evaluation de la `condition`

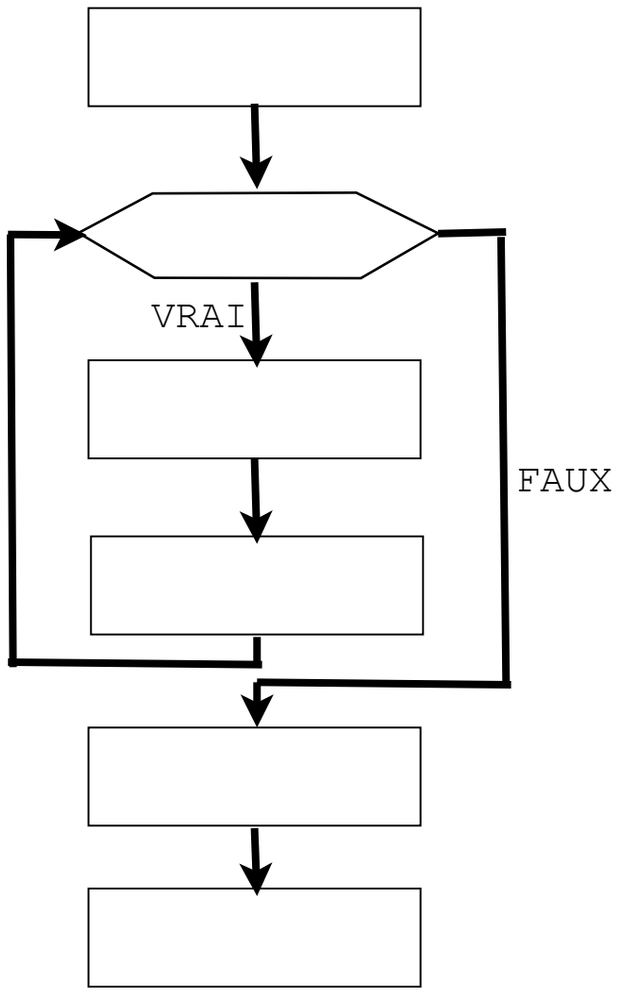
3a. Si `VRAI/true` on **reste** dans la boucle,
c'est-à-dire que l'on revient en 1.

3b. Si `FALSE/false` on **sort** de la boucle

Remarque: le bloc d'instruction est toujours évalué au moins une fois !

# Boucle `TantQue`

- Répète un bloc d'instructions tant qu'une condition est réalisée (ie. `VRAI`)
- A utiliser lorsque l'on ne connaît pas a priori le nombre de tours de boucle
- ATTENTION: la condition est évaluée **en début** de boucle




```
while (<condition>) {
   <bloc d'instructions>
}
```

# Dynamique du `TantQue`

```
while (<condition>) {
   <bloc d'instructions>
}
```

1. Evaluation de la `condition`

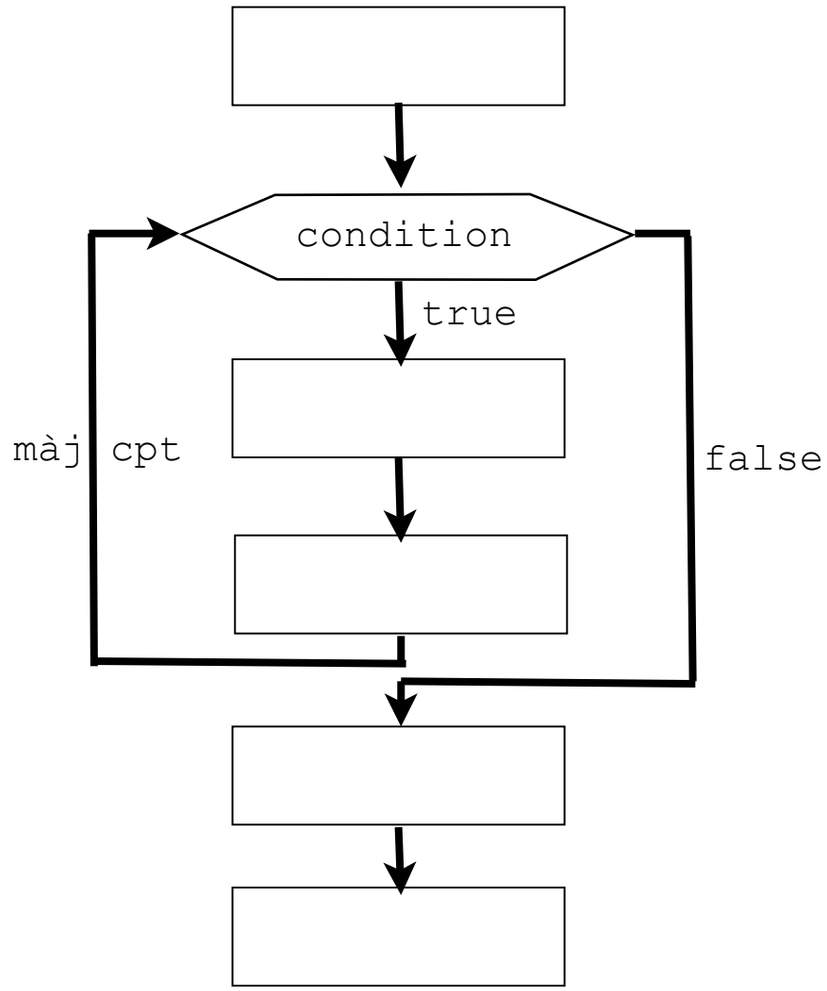
2a. Si `VRAI/true` on **reste** dans la boucle (c'est-à-dire que l'on exécute le corps de la boucle) et l'on revient en 1.

2b. Si `FAUX/false` on **sort** de la boucle

Remarque: il est possible de ne jamais rentrer dans la boucle si la condition est fausse lors de la première évaluation !

# Boucle `Pour`

- Répète un bloc d'instructions un nombre connu de fois
- Utilisation d'un compteur dont la valeur est mise à jour à chaque tour de boucle
- Le compteur évolue entre une borne de début (valeur d'initialisation) et une borne de fin (définie dans la condition)




```
for (<init cpt> ; <condition> ; <màj cpt>) {
   <bloc d'instructions>
}
```

# Dynamique du `Pour`

```
for (<init cpt> ; <condition> ; <màj cpt>) {
   <bloc d'instructions>
}
```

1. Création et initialisation du compteur (`cpt`)
2. Evaluation de la condition (`condition`)

3a. Si la condition est `VRAI/true`, alors:

- le corps de la boucle est exécuté,
- le compteur est mis à jour (`màj cpt`),
- on retourne à l'étape 2.

3b. Si la condition est `FAUX/false`, on sort de la boucle

# Trois types de boucles

```
do {
   <bloc d'instructions>
} while (<condition>) ;
```

```
while (<condition>) {
   <bloc d'instructions>
}
```

Nb de tours inconnus a priori (détection d'évènement)

```
for (<init cpt> ; <condition> ; <màj cpt>) {
   <bloc d'instructions>
}
```

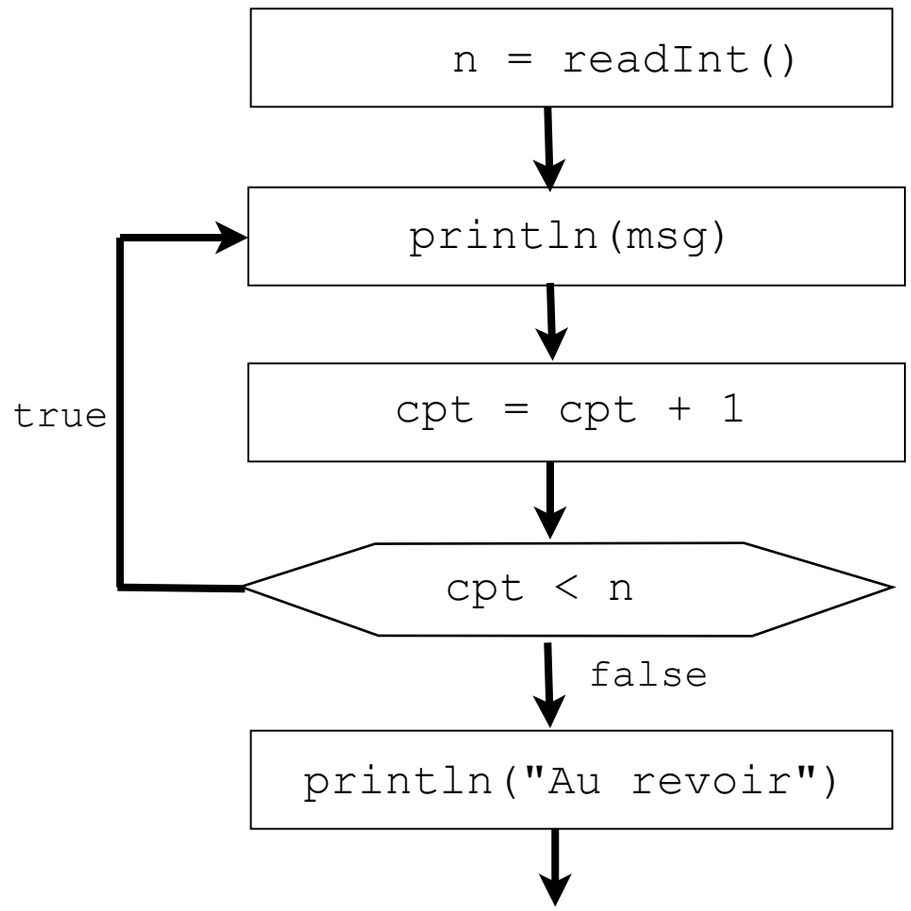
Nb tours connus a priori (boucle à compteur)

# Hello world x `n`

- Illustration des trois types de boucles sur un algorithme simple
- Afficher `n` fois la phrase "`Hello World`"
- Gestion manuelle du compteur de tour pour les boucles `Répéter` et `TantQue`
- Version plus compacte avec la boucle `Pour`

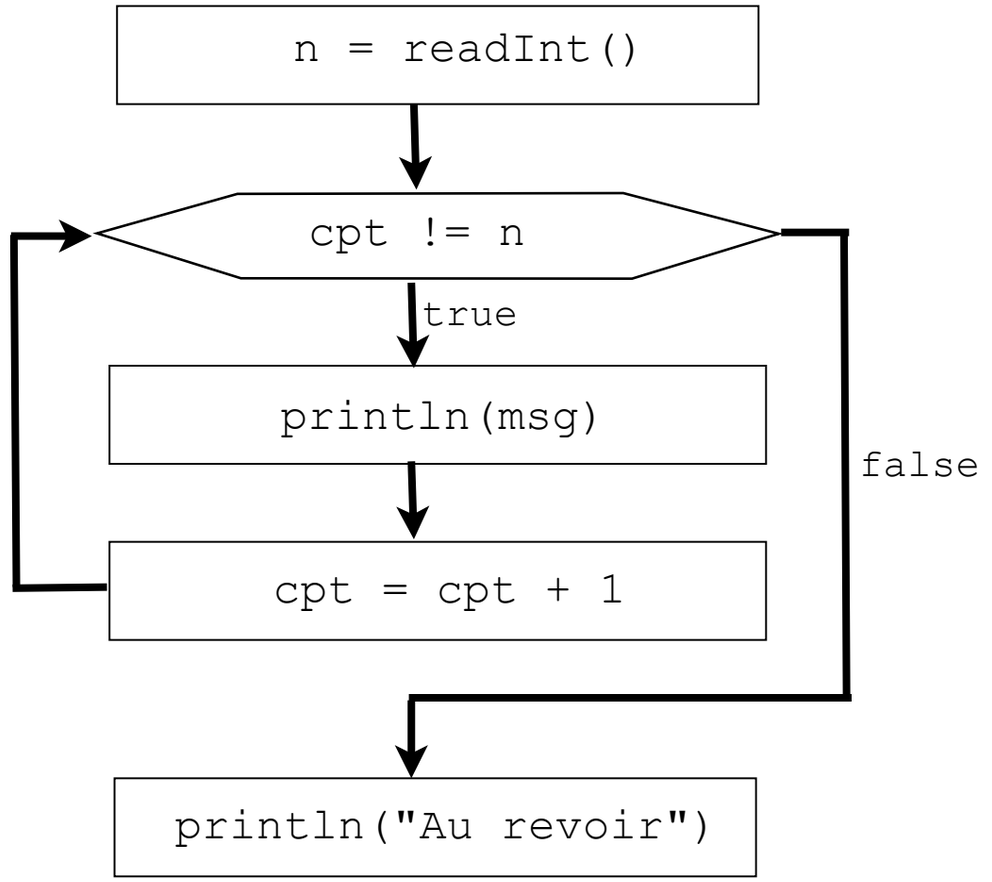
# Répéter **Hello World**

```
class HelloWorldRepeter extends Program {
  void algorithm() {
    String msg = "Hello World";
    int n, cpt = 0;

    n = readInt();
    do {
      println(msg);
      cpt = cpt + 1;
    } while (cpt < n);
    println("Au revoir");
  }
}
```



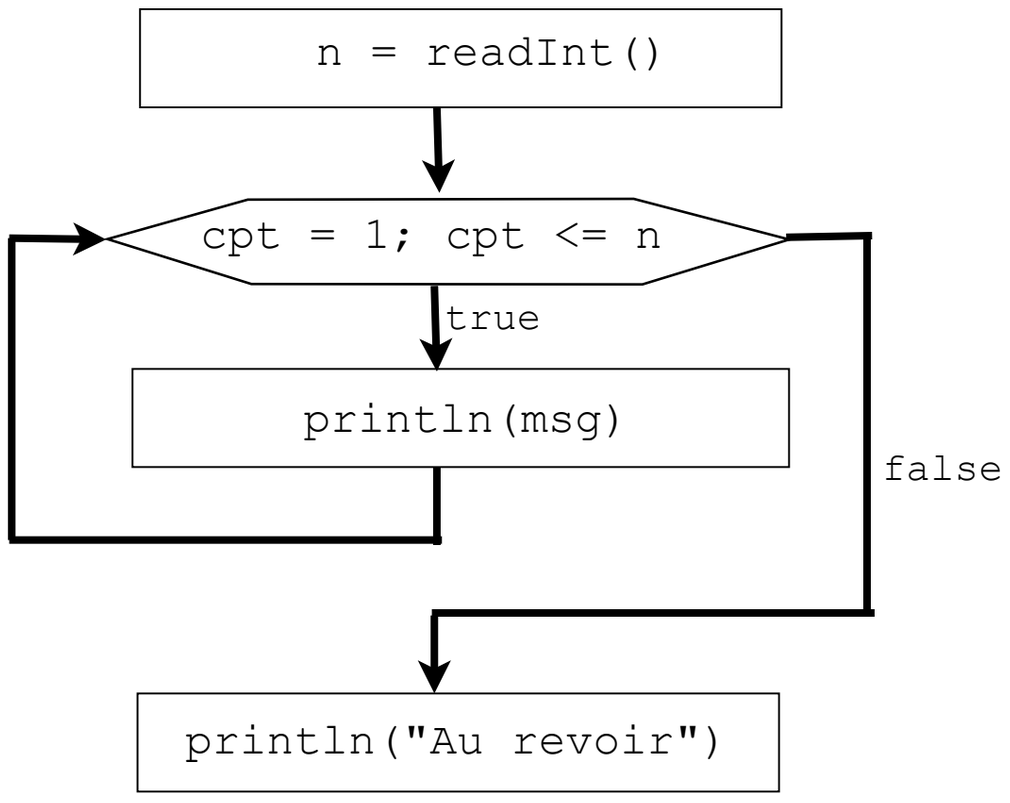
# TantQue Hello World

```
class HelloWorldTantQue extends Program {
  void algorithm() {
    String msg = "Hello World";
    int n, cpt = 0;

    n = readInt();
    while (cpt != n) {
      println(msg);
      cpt = cpt + 1;
    }
    println("Au revoir");
  }
}
```



# Pour **Hello World**

```
class HelloWorldPour extends Program {
  void algorithm() {
    String msg = "Hello World";
    int n;

    n = readInt();
    for (int cpt=1; cpt<=n; cpt=cpt+1) {
      println(msg);
    }
    println("Au revoir");
  }
}
```



# Comment choisir ?

- Nombre de tours connus: `Pour`
- Sinon généralement `TantQue` et plus rarement `Répéter`
- Attention aux boucles infinies avec les boucles à détection d'évènement
- Toujours tester l'entrée et la sortie de boucle !

# Algorithme du perroquet

- Concevoir un algorithme qui demande une chaîne à l'utilisateur et la répète, jusqu'à ce que l'utilisateur entre `"STOP"`
- Nécessite une boucle:
  - Quelle séquence d'instructions est répétée ?
  - Quelle est la condition de sortie de boucle ?

# Exemple d'exécution

Entrez une phrase: **Bonjour Hal**

***Bonjour Hal*** *// Hi Dave.*

Entrez une phrase: **Ca va ?**

***Ca va ?***

*// I am completely operational, and all my circuits are functionning perfectly.*

Entrez une phrase: **Tu en es sûr ?**

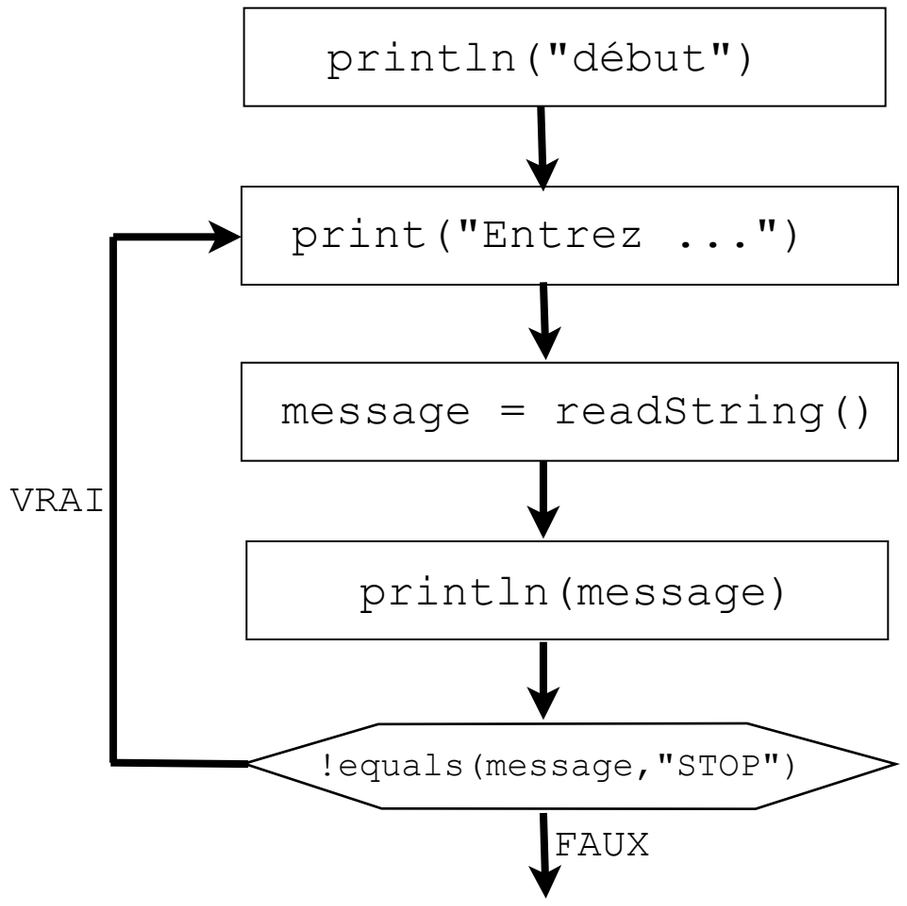
**Tu en es sûr ?**

*// I am putting myself to the fullest possible use, which is all I think that any conscious entity can ever hope to do.*

Entrez une phrase: **STOP**

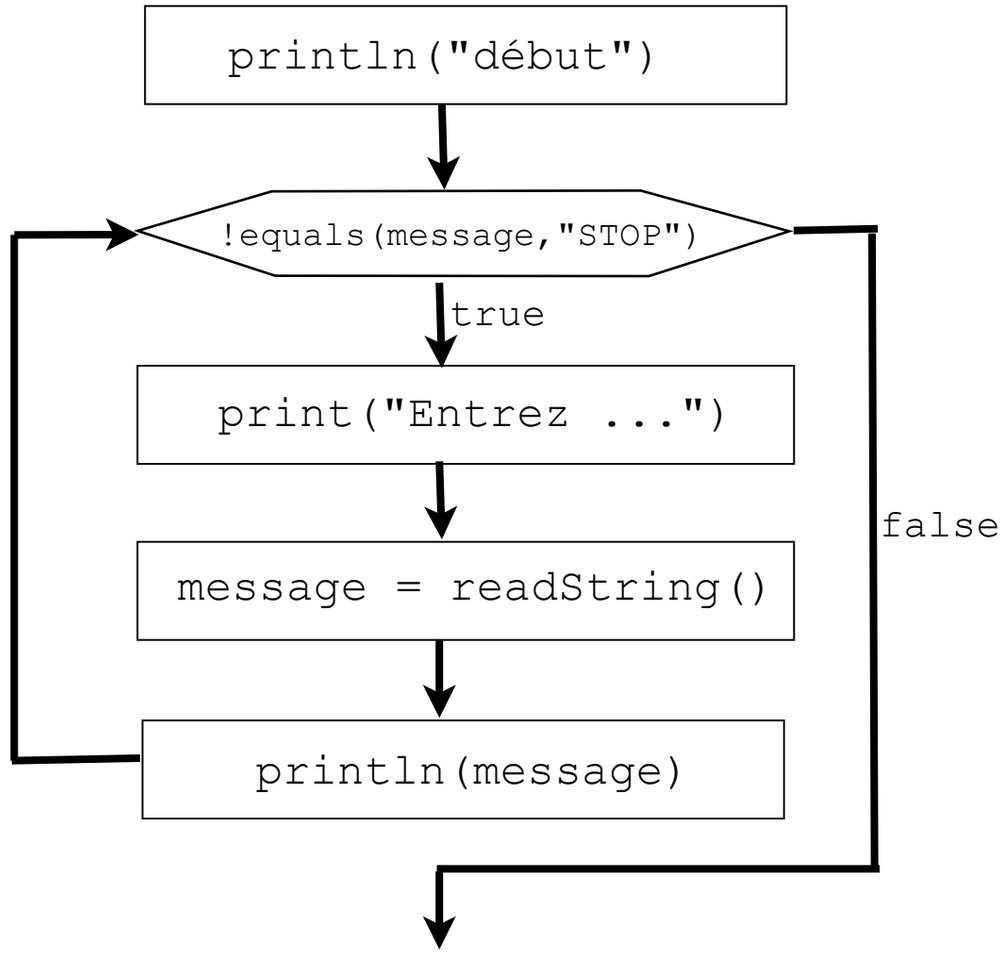
# Algorithme Perroquet

```
class Perroquet extends Program {
  void algorithm() {
    String message;

    println("début");
    do {
      println("Entrez une phrase: ");
      message = readString();
      println(message);
    } while (!equals(message,"STOP"));
  }
}
```



# Algorithme Perroquet

```
class Perroquet extends Program {
  void algorithm() {
    String message = "";

    println("début")
    while (!equals(message,"STOP")) {
      print("Entrez une phrase: ");
      message = readString();
      println(message)
    }
  }
}
```



# Slide ajouté durant le cours …

```
class Perroquet extends Program {
  void algorithm() {
    String message;

    print("Entrez une phrase: ");
    message = readString();

    while (!equals(message,"STOP")) {
      println(message)
      print("Entrez une phrase: ");
      message = readString();
    }
  }
}
```

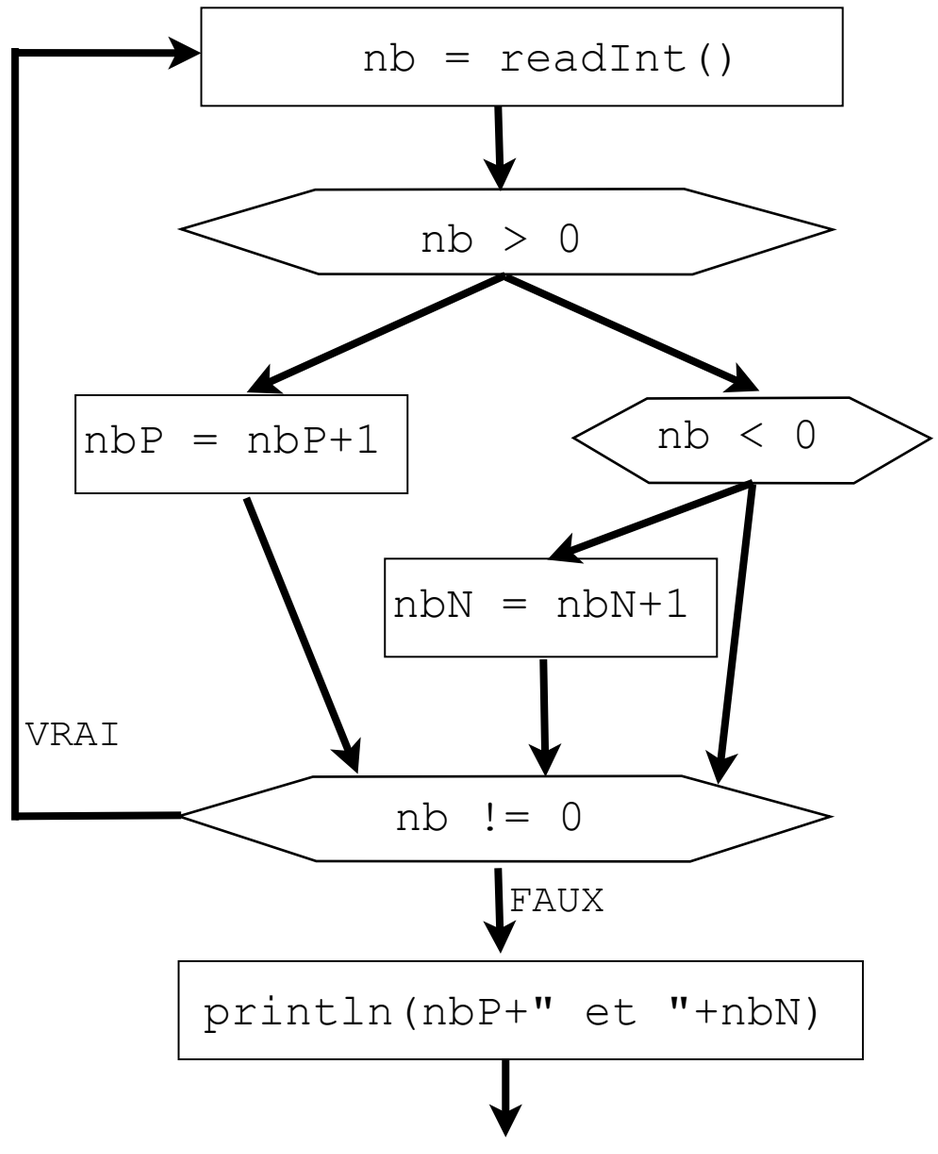
```
class Perroquet extends Program {
  void algorithm() {
    String message = "";

    while (!equals(message,"STOP")) {
      println(message);
      print("Entrez une phrase: ");
      message = readString();
    }
  }
}
```

# Décompte de nombres

- Décompter le nombre de nombres positifs et négatifs dans une série de nombre
- La saisie de nombre se poursuit tant que l'utilisateur n'entre pas le nombre 0
- Type de boucle ?

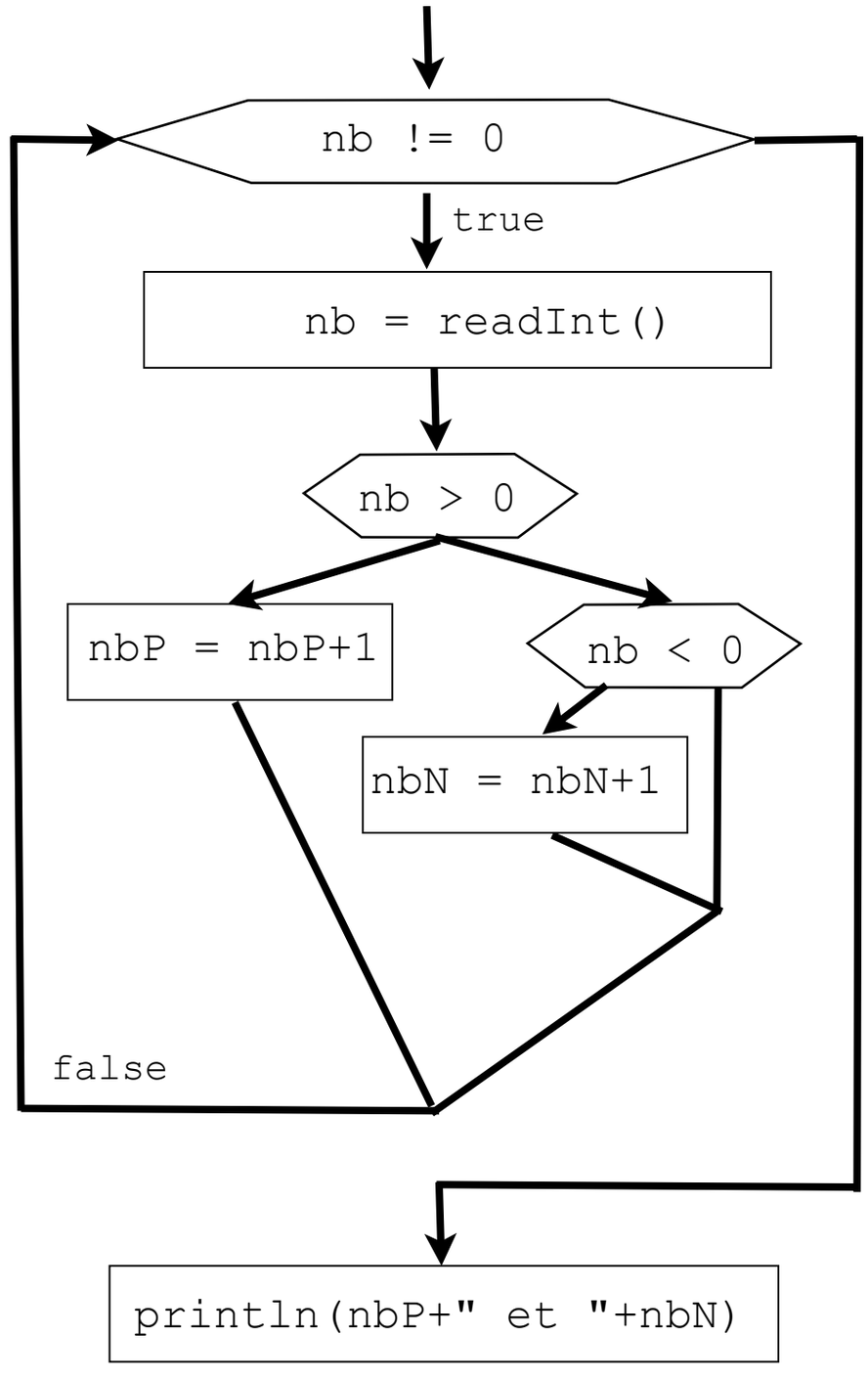
# Décompte de nombres

```
class CompterPosNeg extends Program {
  void algorithm() {
    int nbPositif = 0, nbNegatif = 0;
    int nb;

    do {
      nb = readInt();
      if (nb > 0) {
        nbPositif = nbPositif + 1;
      } else if (nb < 0) {
        nbNegatif = nbNegatif + 1;
      }
    } while (nb != 0);
    println(nbPositif + " et " + nbNegatif);
  }
}
```



# Décompte de nombres

```
class CompterPositifNegatif extends Program {
  void algorithm() {
    int nbPositif = 0, nbNegatif = 0;
    int nb = ?;

    while (nb != 0) {
      nb = readInt();
      if (nb > 0) {
        nbPositif = nbPositif + 1;
      } else if (nb < 0) {
        nbNegatif = nbNegatif + 1;
      }
    }
    println(nbPositif + " et " + nbNegatif);
  }
}
```



# Calcul d'une factorielle

- Concevoir un algorithme calculant la factorielle d'un nombre
- Définition: `0!=1` et `n!=1x2x...xn`
- Première version: `dowhile`
- Deuxième version: `for`

# Factorielle avec `dowhile`

```
class Factorielle extends Program {
    void testAssertFactorielleDoWhile() {
        assertEquals(  1, factorielleDoWhile(1));
        assertEquals(  2, factorielleDoWhile(2));
        assertEquals(  6, factorielleDoWhile(3));
        assertEquals( 24, factorielleDoWhile(4));
        assertEquals(120, factorielleDoWhile(5));
    }
    int factorielleDoWhile(int n) {
        int factorielle = 1;
        int cpt = n;
        do {
            factorielle = factorielle * cpt;
            cpt = cpt - 1;
        } while (cpt != 0);
        return factorielle;
    }
}
```

# Factorielle avec dowhile

```
int factorielleDoWhile(int n) {
    int factorielle = 1;
    int cpt = n;
    if (n>1) {
        do {
            factorielle = factorielle * cpt;
            cpt = cpt - 1;
        } while (cpt != 0);
    }
    return factorielle;
}
```

# Factorielle avec `for`

```
int factorielleFor(int n) {
    int factorielle = 1;
    for (int cpt = 2; cpt <= n; cpt = cpt+1){
        factorielle = factorielle * cpt;
    }
    return factorielle;
}
```

# Moyenne de 5 notes

- Concevoir un algorithme calculant la moyenne de 5 notes
- Deux versions: `Répéter` et `Pour`

# Version Répéter

```
class Moyenne extends Program {
  int moyenne() {
    int nbNotes = 0, note;
    int somme = 0, moyenne;
    do {
      note = readInt();
      somme = somme + note;
      nbNotes = nbNotes + 1;
    } while (nbNotes < 5);
    return somme / nbNotes;
  }
}
```

```
class Moyenne extends Program {
  int moyenne() {
    int nbNotes = 0, note;
    int somme = 0, moyenne;
    do {
      note = readInt();
      if (note>=0 && note <=20) {
        somme = somme + note;
        nbNotes = nbNotes + 1;
      }
    } while (nbNotes < 5);
    return somme / nbNotes;
  }
}
```

# Version Pour

```
class Moyenne extends Program {
  int moyenne() {
    int nbNotes = 5, somme = 0;
    int note, moyenne;

    for (int i=0; i<nbNotes;i=i+1){
      note = readInt();
      if (note>=0 && note<=20) {
        somme = somme + note;
      }
    }
    return somme / nbNotes;
  }
}
```

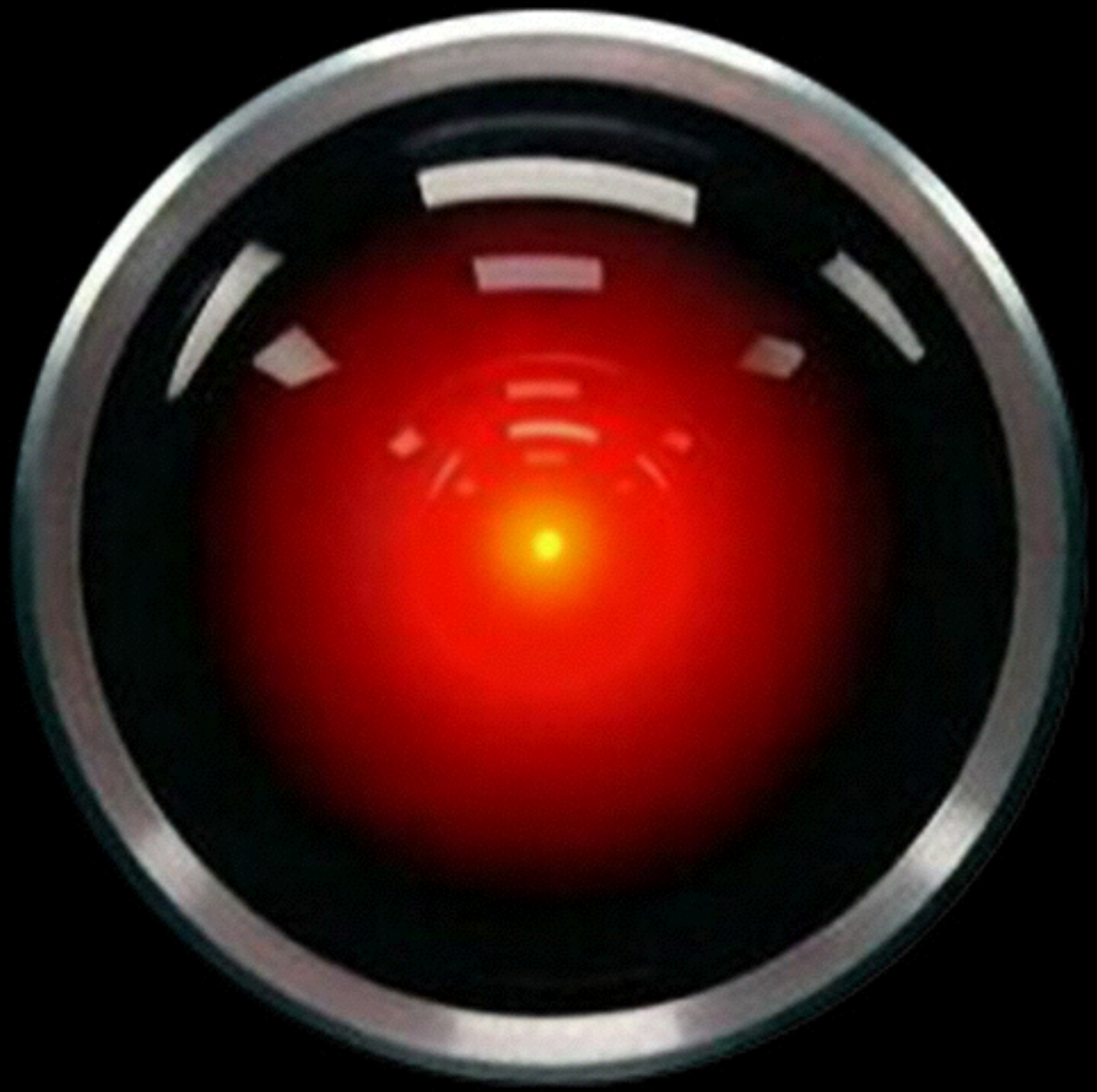
```
class Moyenne extends Program {
  int moyenne() {
    int nbNotes = 5, somme = 0;
    int note, moyenne;

    for (int i=0; i<nbNotes;i=i+1){
      do {
        note = readInt();
      } while (note<0 || note>20);
      somme = somme + note;
    }
    return somme / nbNotes;
  }
}
```

# Notes éliminatoires

- Saisir 5 notes au maximum sauf si une note éliminatoire est rencontrée
- Note éliminatoire si inférieure à 5
- Deux évènements: 5 notes / note éliminatoire
- Utilisation de variables booléennes (aussi appelées drapeaux)

```
class CalculMoyenne extends Program {
  int moyenne() {
    int nbNotes = 0, somme = 0;
    int note, moyenne;
    boolean fini = false, noteEliminatoire = false;
    while (!fini ?? !noteEliminatoire) {
      note = readInt();
      if (note >= 0 && note < 5) {
        noteEliminatoire = true;
      } else if (note >= 5 && note <= 20) {
        somme = somme + note;
        nbNotes = nbNotes + 1;
        if (nbNotes == 5) {
          fini = true;
        }
      }
    }
    if (noteEliminatoire) {
      return -1;
    } else {
      return somme / nbNotes ;
    }
  }
}
```

?



# Chaînes et boucles

- Parcourir une chaîne est courant
- Le parcours peut se faire dans les 2 sens
- Cas d'utilisation classique du `Pour`
- Ex: afficher les lettres d'une chaînes

# Jeu de miroir

- L'utilisateur saisit un mot et l'on construit le mot miroir (ie. lettres inversées)
- Type de boucle la plus adaptée ?
- Quatre solutions possibles !