

# Algorithmique & Programmation

## Notion de répétition

[yann.secq@univ-lille.fr](mailto:yann.secq@univ-lille.fr)

ABDELKADER Omar, BIRLOUEZ Martin, BONEVA Iovka, DELECROIX Fabien, LEQUINIOU Erwann, MARSHALL-BRETON Christopher, REKIK Yosra, SECQ Yann, SOW Younoussa, SUDHEENDRAN Megha, SUE Yue

# Structure de contrôle

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

# Structures de contrôle

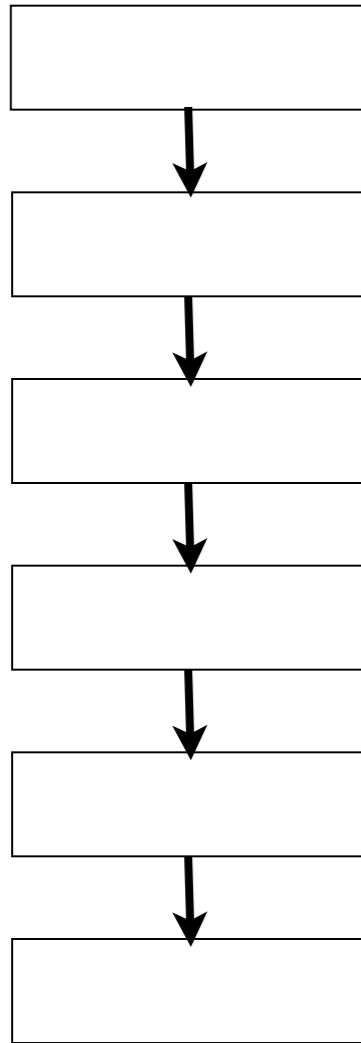
- Les alternatives permettent des branchements
- Les boucles permettent des répétitions
- **Boucle : répéter un bloc d'instructions un certain nombre de fois (dénombrable a priori ou pas)**

Variables + Alternative + Boucle + Fonction

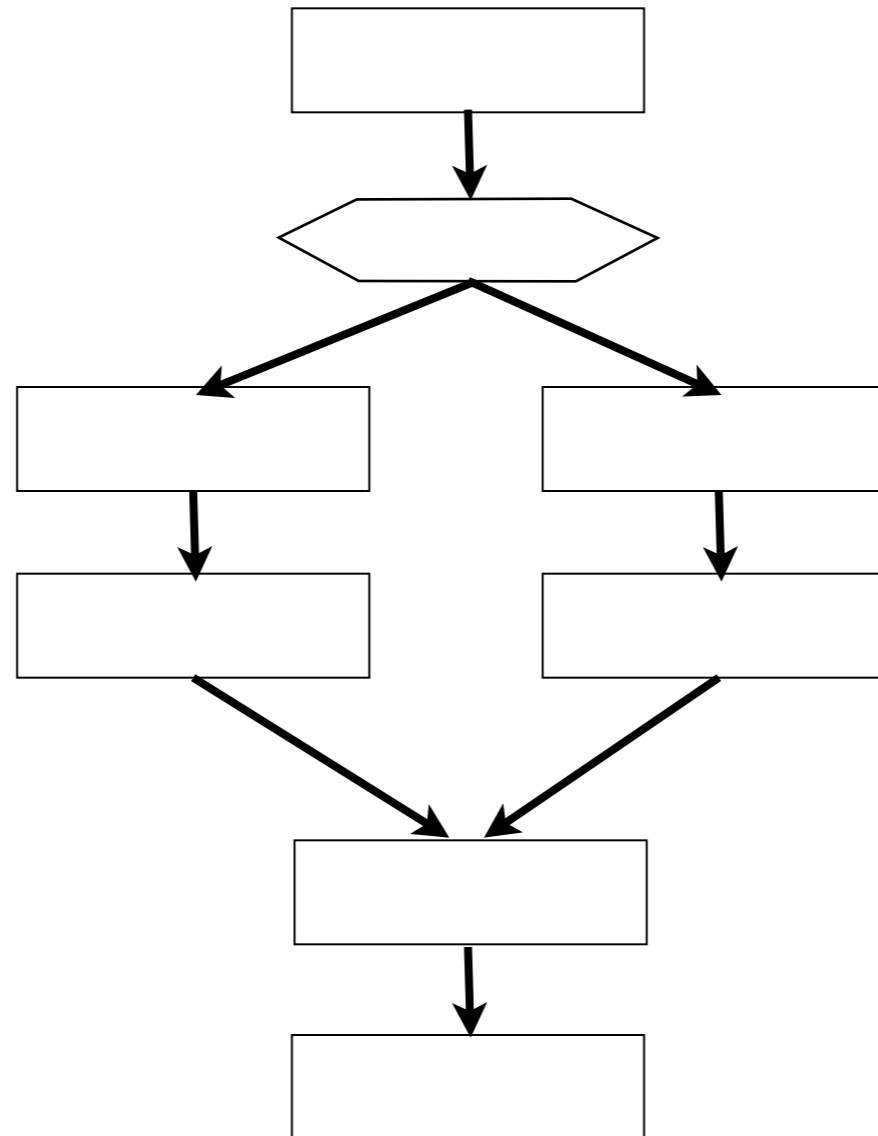
=

Bases de l'algorithmique

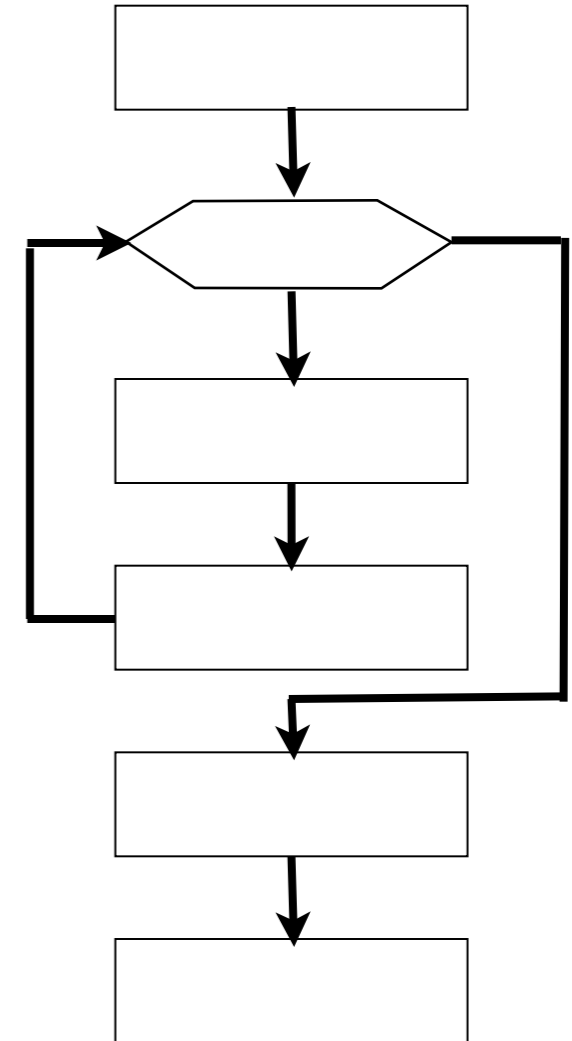
# Structures de contrôle



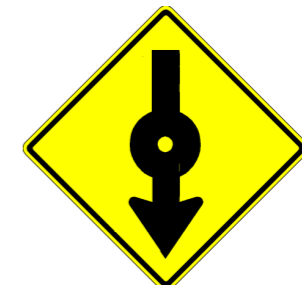
Séquence



Alternative



Répétition



# Deux familles de répétition

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

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

Nombre d'itérations inconnu a priori  
**boucle à évènement**

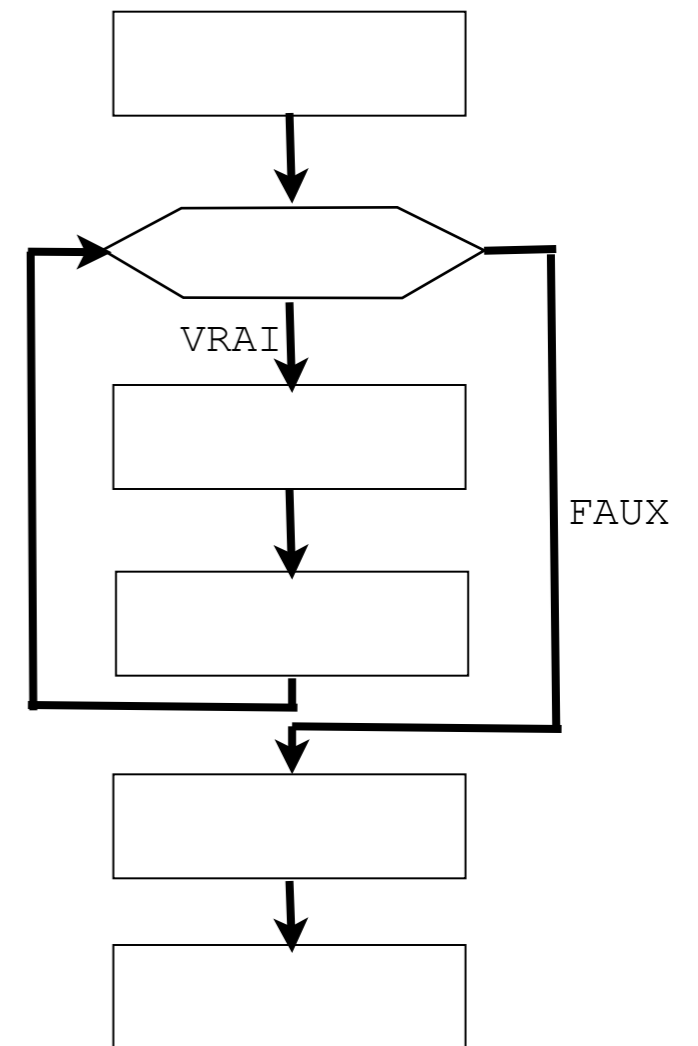
```
for (<init cpt> ; <condition> ; <màj cpt>) {  
    <bloc d'instructions>  
}
```

Nombre d'itérations connu a priori  
**boucle à compteur**

# 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>  
}
```



# Évaluation 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.

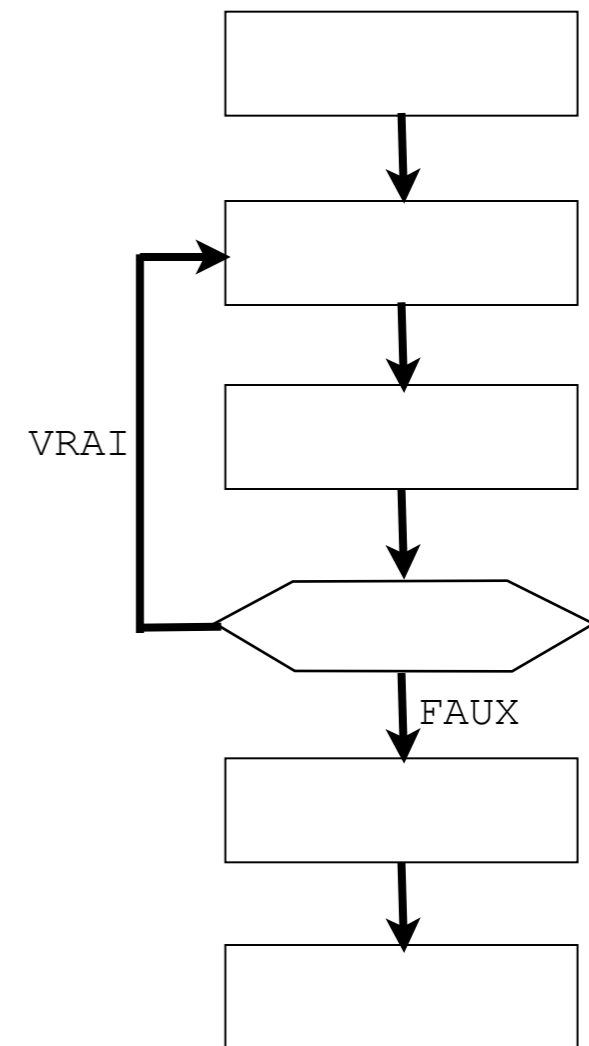
2b. Si `FAUX/false` on **sort** de la boucle

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

# Répéter TantQue

- 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 et que l'on souhaite au moins une exécution du corps de la boucle**
- ATTENTION: la condition est évaluée **en fin** de boucle

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



# Évaluation du Répéter TantQue

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

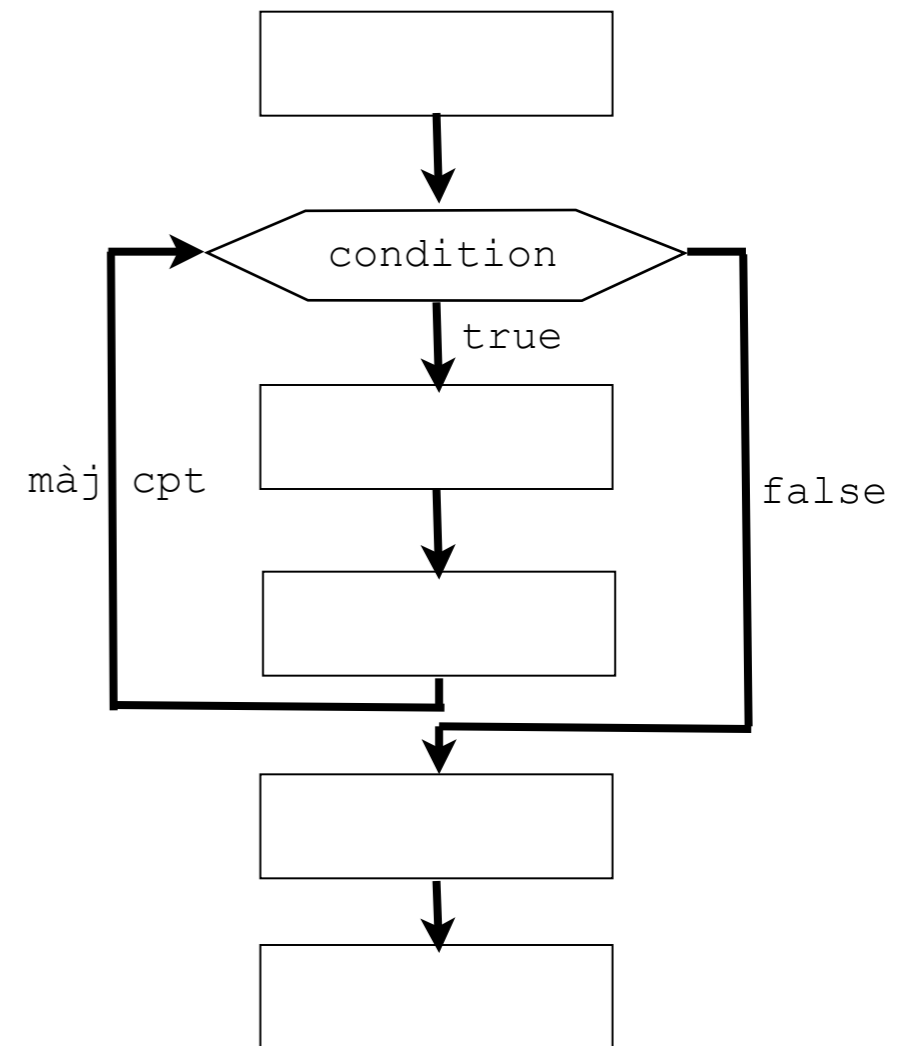
1. Le bloc d'instruction est évalué
2. Evaluation de la condition
- 3a. Si VRAI/true on **reste** dans la boucle,  
c'est-à-dire que l'on revient en 1.
- 3b. Si FALSE/false on **sort** de la boucle

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

# Pour

- Répète un bloc d'instructions un nombre connu de fois
- **A utiliser lorsque l'on connaît a priori le nombre de tours de boucle**
- 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>  
}
```



# Évaluation 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

# Deux formes de boucles

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

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

Nb de tours inconnus a priori : **boucle à évènement**

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

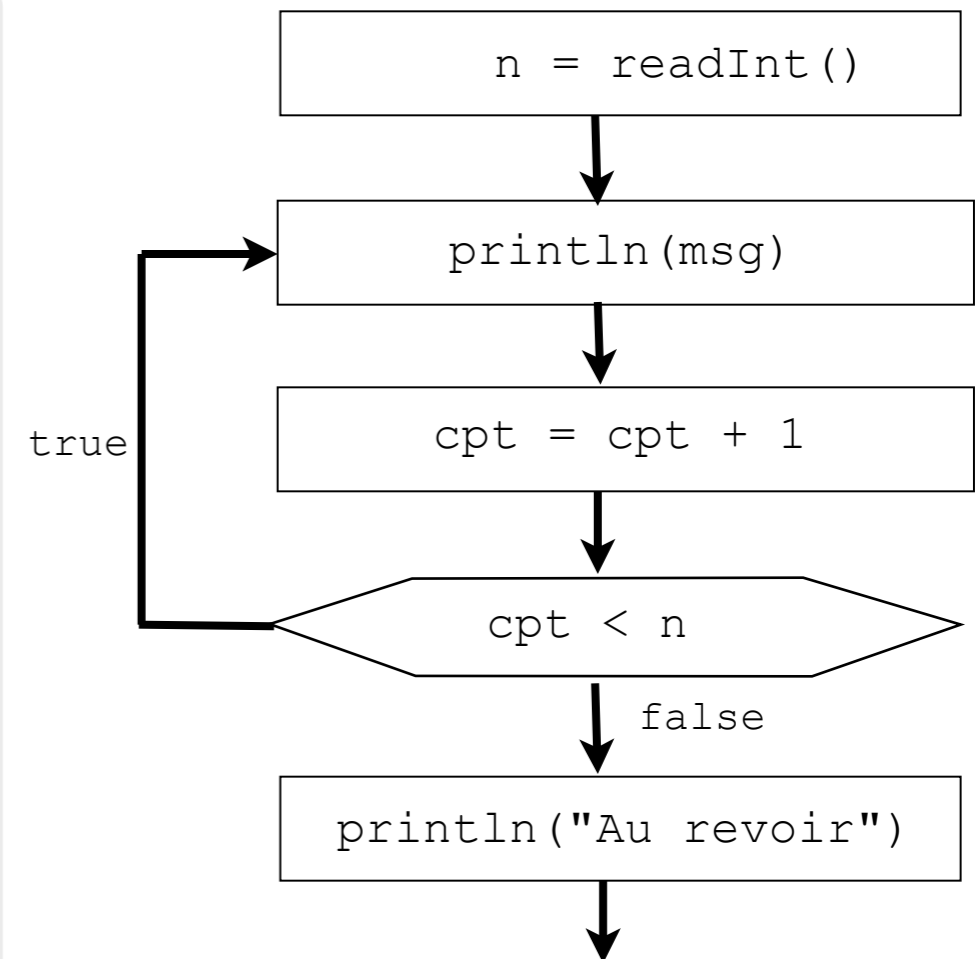
Nb de tours connus a priori : **boucle à compteur**

# Hello world x n

- Illustration des trois types de boucles sur un algorithme simple
- Afficher  $n$  fois la phrase "Hello World"
- Gestion complète du compteur de tour pour les boucles RépéterTantQue et TantQue
- Version plus compacte avec la boucle Pour avec l'incrémentation du compteur déclenchée automatiquement en fin d'itération

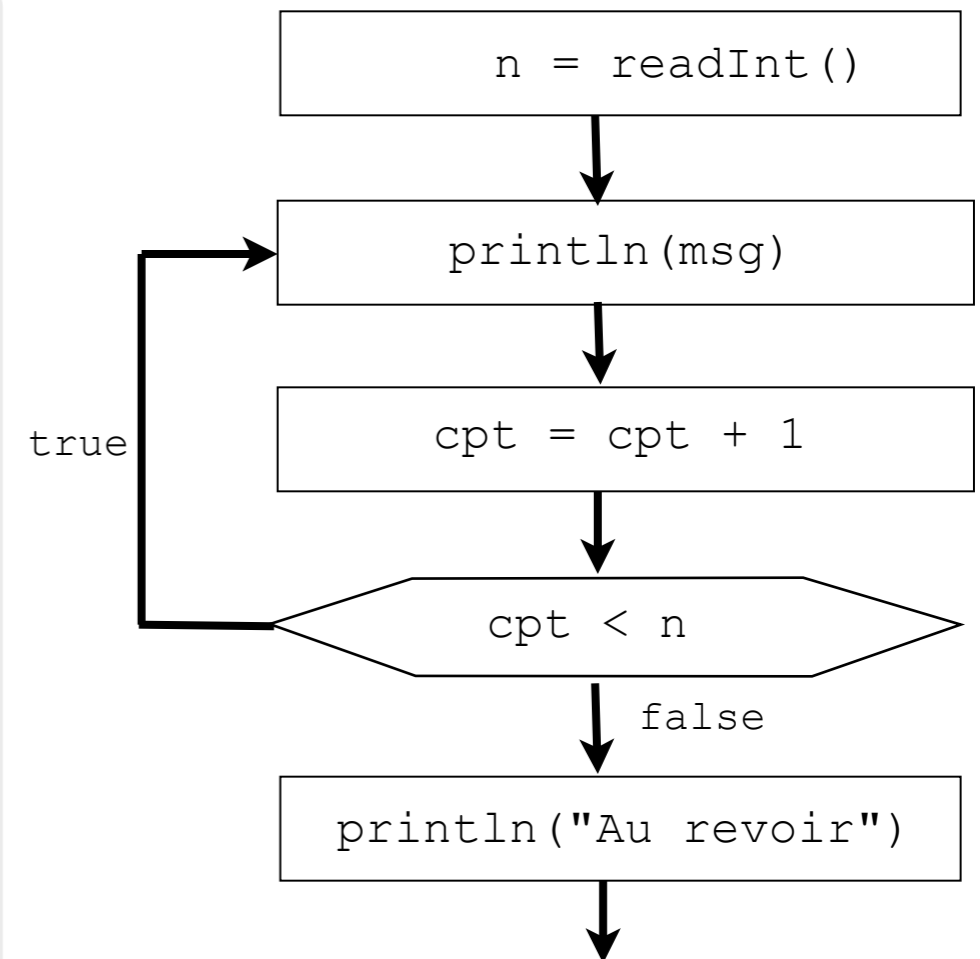
# Répéter Tant Que Hello World

```
class HelloWorldRepeater extends Program {  
    void algorithm() {  
        final String MESSAGE = "Hello World";  
        int n, cpt = 0;  
  
        n = readInt();  
        do {  
            println(MESSAGE);  
            cpt = cpt + 1;  
        } while (cpt < n);  
        println("Au revoir");  
    }  
}
```



# Répéter Tant Que Hello World

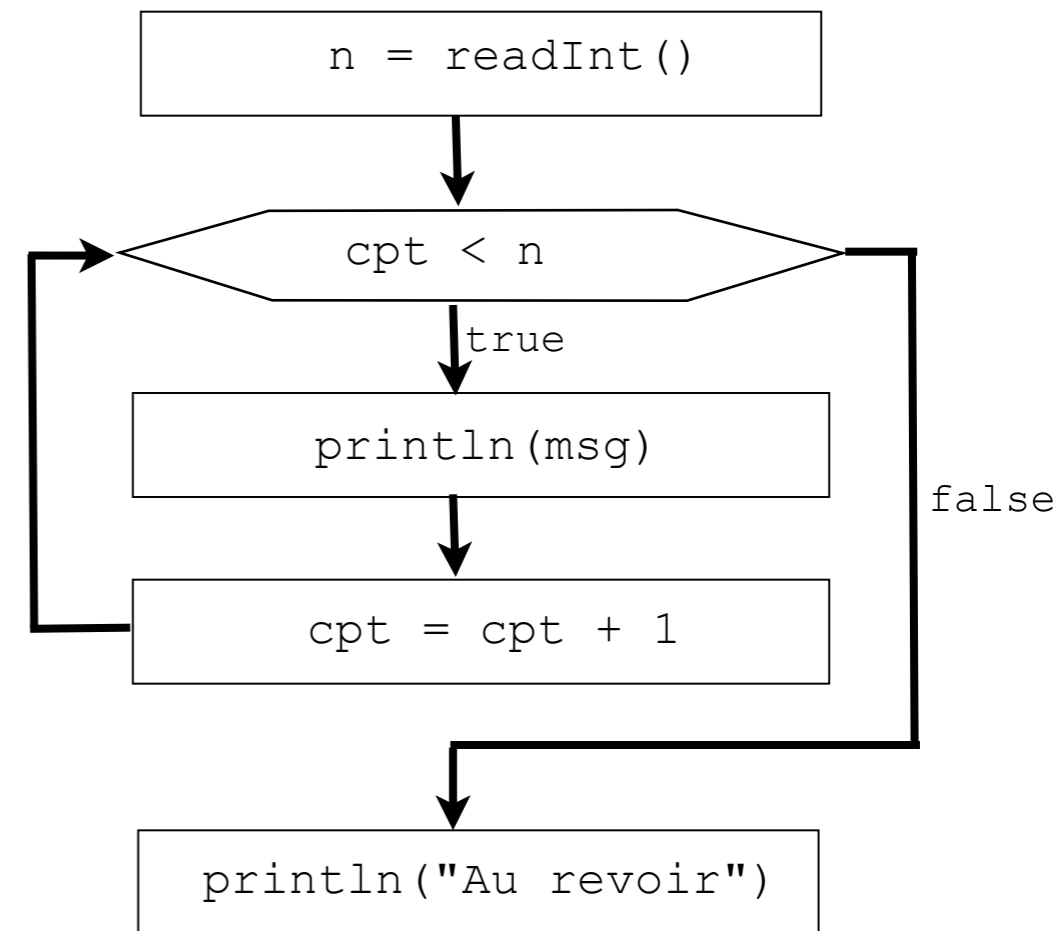
```
class HelloWorldRepeater extends Program {  
    void algorithm() {  
        final String MESSAGE = "Hello World";  
        int n, cpt = 0;  
  
        n = readInt();  
        do {  
            println(MESSAGE);  
            cpt = cpt + 1;  
        } while (cpt < n);  
        println("Au revoir");  
    }  
}
```



*Attention : toujours au moins un affichage !*

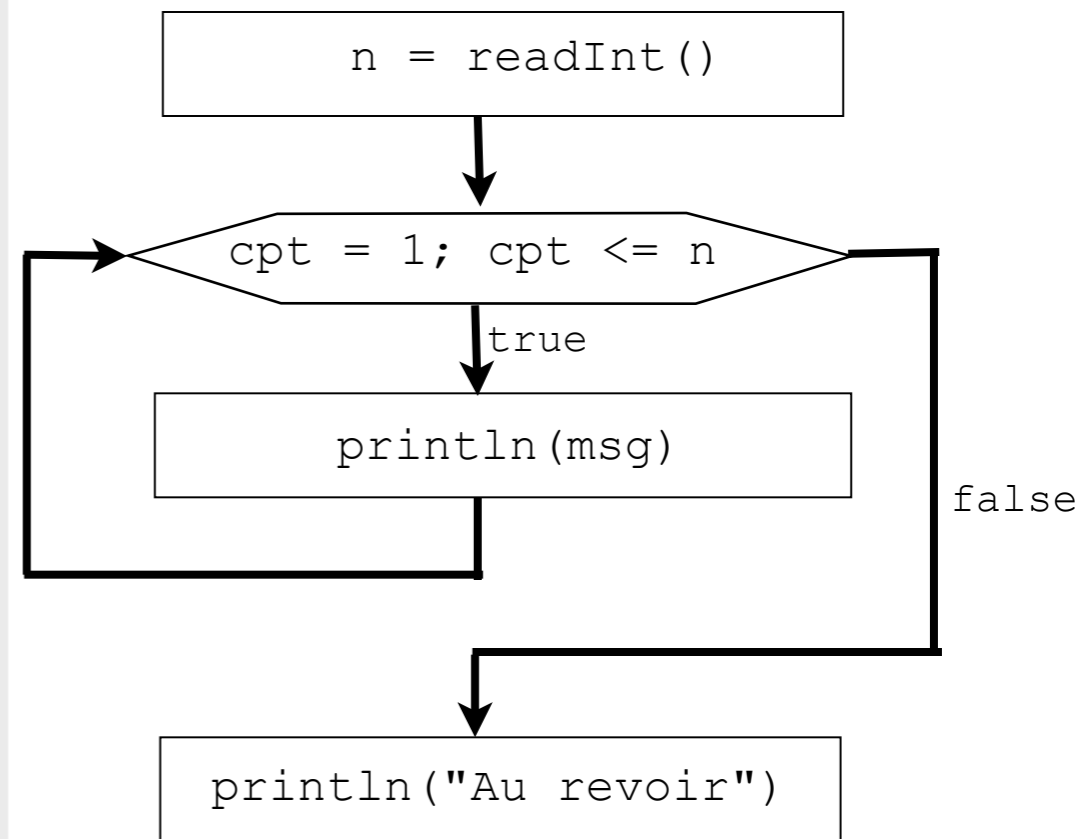
# TantQue Hello World

```
class HelloWorldTantQue extends Program {  
    void algorithm() {  
        final String MESSAGE = "Hello World";  
        int n, cpt = 0;  
  
        n = readInt();  
        while (cpt < n) {  
            println(MESSAGE);  
            cpt = cpt + 1;  
        }  
        println("Au revoir");  
    }  
}
```



# Pour Hello World

```
class HelloWorldPour extends Program {  
    void algorithm() {  
        final String MESSAGE = "Hello World";  
        int n;  
  
        n = readInt();  
        for (int cpt=1; cpt<=n; cpt=cpt+1) {  
            println(MESSAGE);  
        }  
        println("Au revoir");  
    }  
}
```



*Que se passe-t-il pour  $n=0$  ?*

# Comment choisir ?

- Nombre d'itérations connus a priori : `Pour`
- Sinon quasiment toujours `TantQue` et plus rarement `RépéterTantQue` dès que l'on souhaite au l'exécution du corps de boucle une fois
- **Toujours tester l'entrée et la sortie de boucle !**
- **Attention aux boucles infinies avec les boucles à détection d'évènement**

# Algorithme du perroquet

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

# Exemple d'exécution

Entrez une phrase: **Bonjour Hal**

***Bonjour Hal***

Entrez une phrase: **Ca va ?**

***Ca va ?***

Entrez une phrase: **Tu en es sûr ?**

***Tu en es sûr ?***

Entrez une phrase: **STOP**

*// Hi Hal*

*// Hi Dave*

*// Are you alright Hal ?*

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

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

# Exemple d'exécution

Entrez une phrase: **Bonjour Hal**

***Bonjour Hal***

Entrez une phrase: **Ca va ?**

***Ca va ?***

Entrez une phrase: **Tu en es sûr ?**

***Tu en es sûr ?***

Entrez une phrase: **STOP**

*// Hi Hal*

*// Hi Dave*

*// Are you alright Hal ?*

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

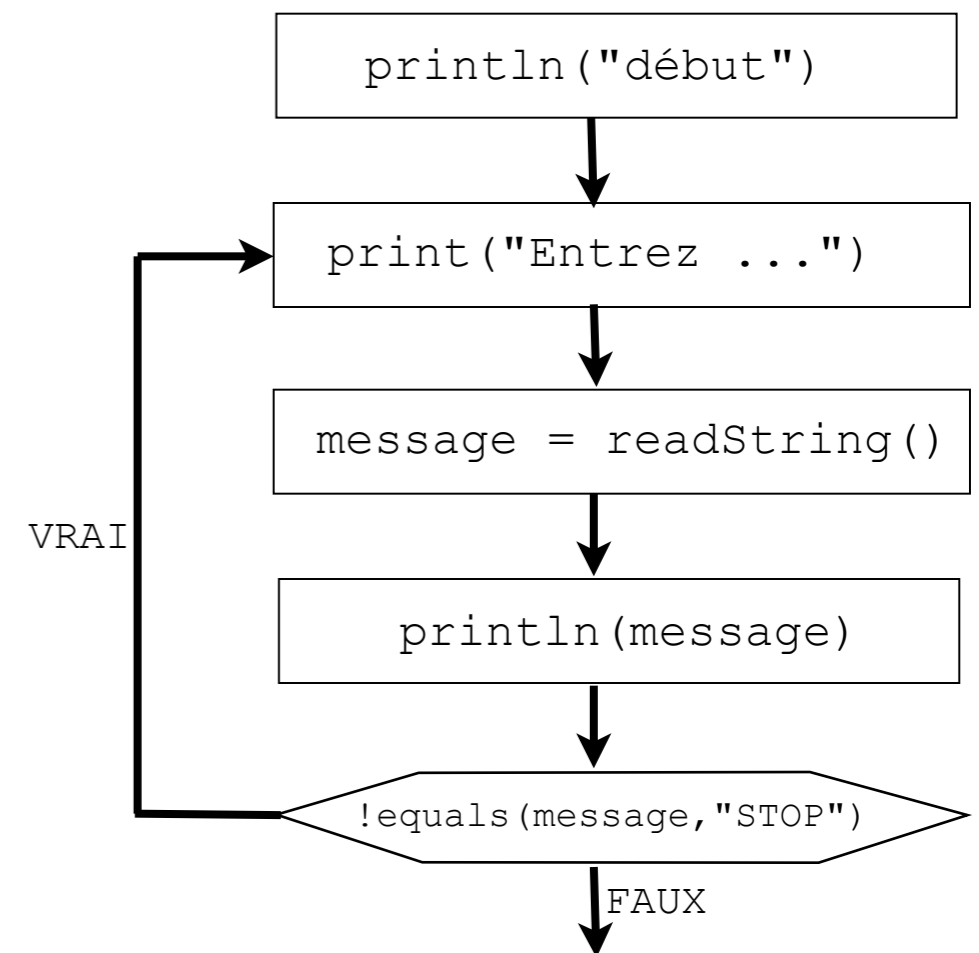
*// I am putting myself to the  
fullest possible use, which is  
all I think any conscious  
entity can ever hope to do.*

***ChatGPT ?***

***Suivez le conseil d'Hal,  
mobilisez d'abord votre  
cerveau !***

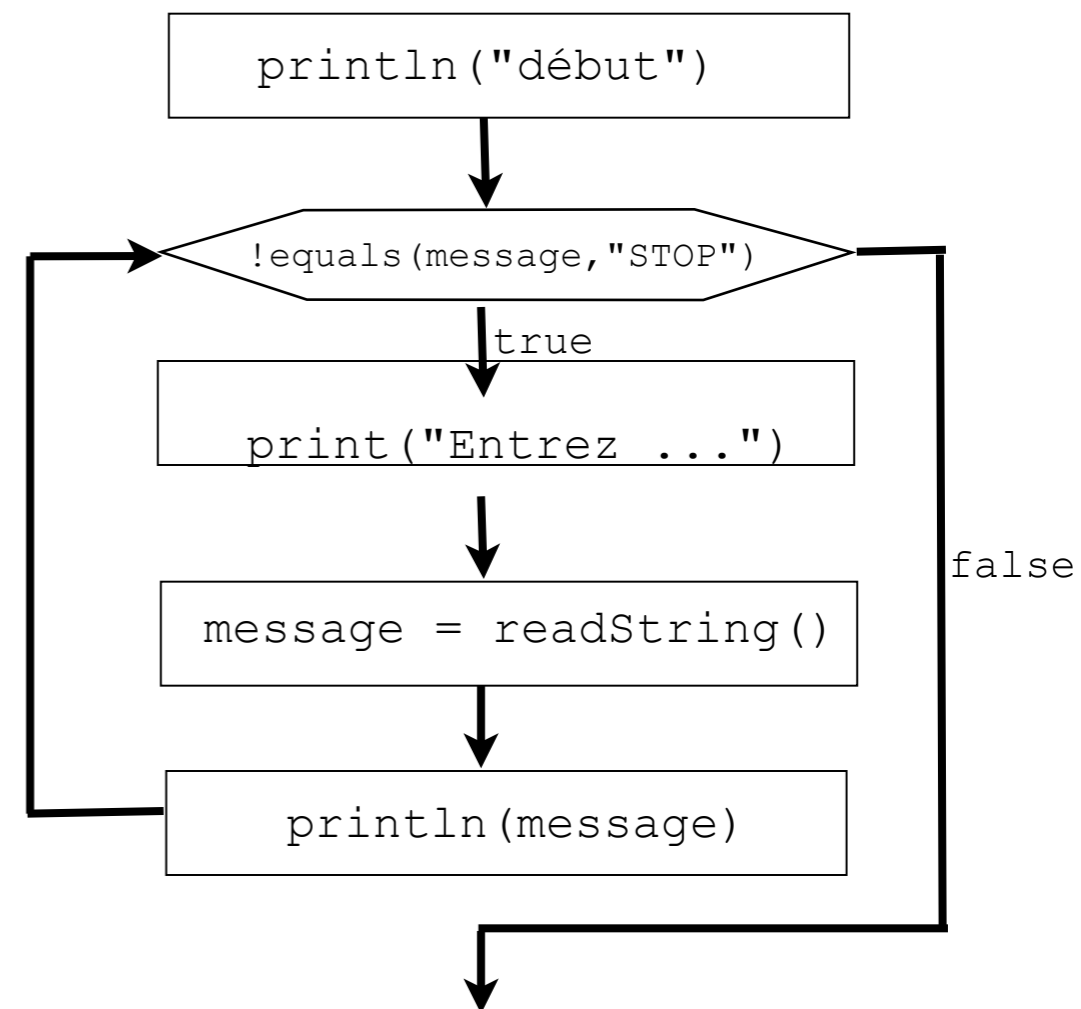
# Algorithme Perroquet

```
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 un précédent 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();  
        }  
    }  
}
```

*Redondance de code*

```
class Perroquet extends Program {  
    void algorithm() {  
        String message = "";  
  
        while (!equals(message, "STOP")) {  
            println(message);  
            print("Entrez une phrase: ");  
            message = readString();  
        }  
    }  
}
```

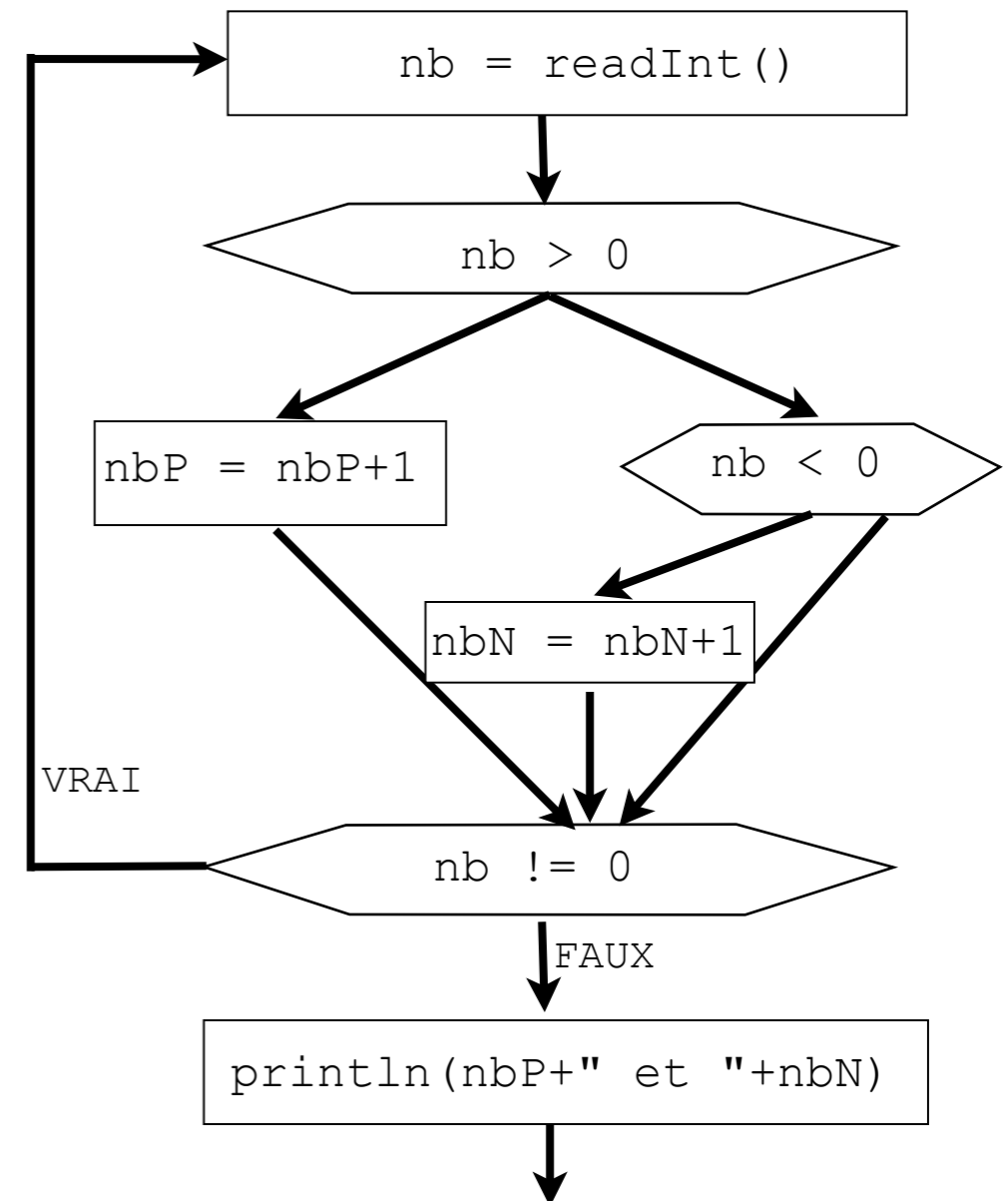
*Hack visible par l'utilisateur ...*

# Décompte de nombres

- Décompter les nombres positifs et négatifs dans une série de nombres saisis au clavier
- La saisie des nombres se poursuit tant que l'utilisateur n'entre pas le nombre 0
- Quel type de boucle mobiliser ?

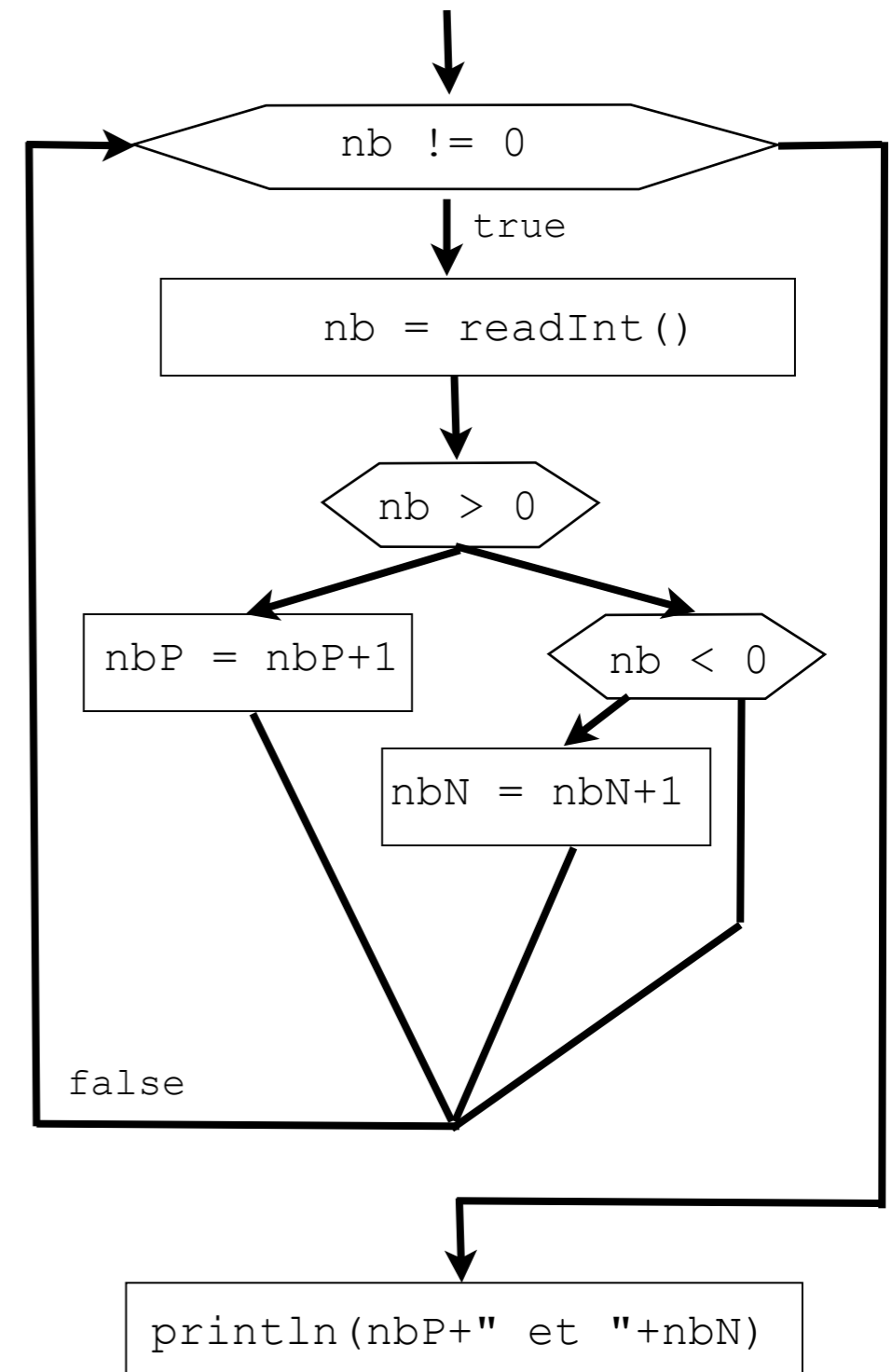
# Décompte de nombres

```
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);  
    }  
}
```



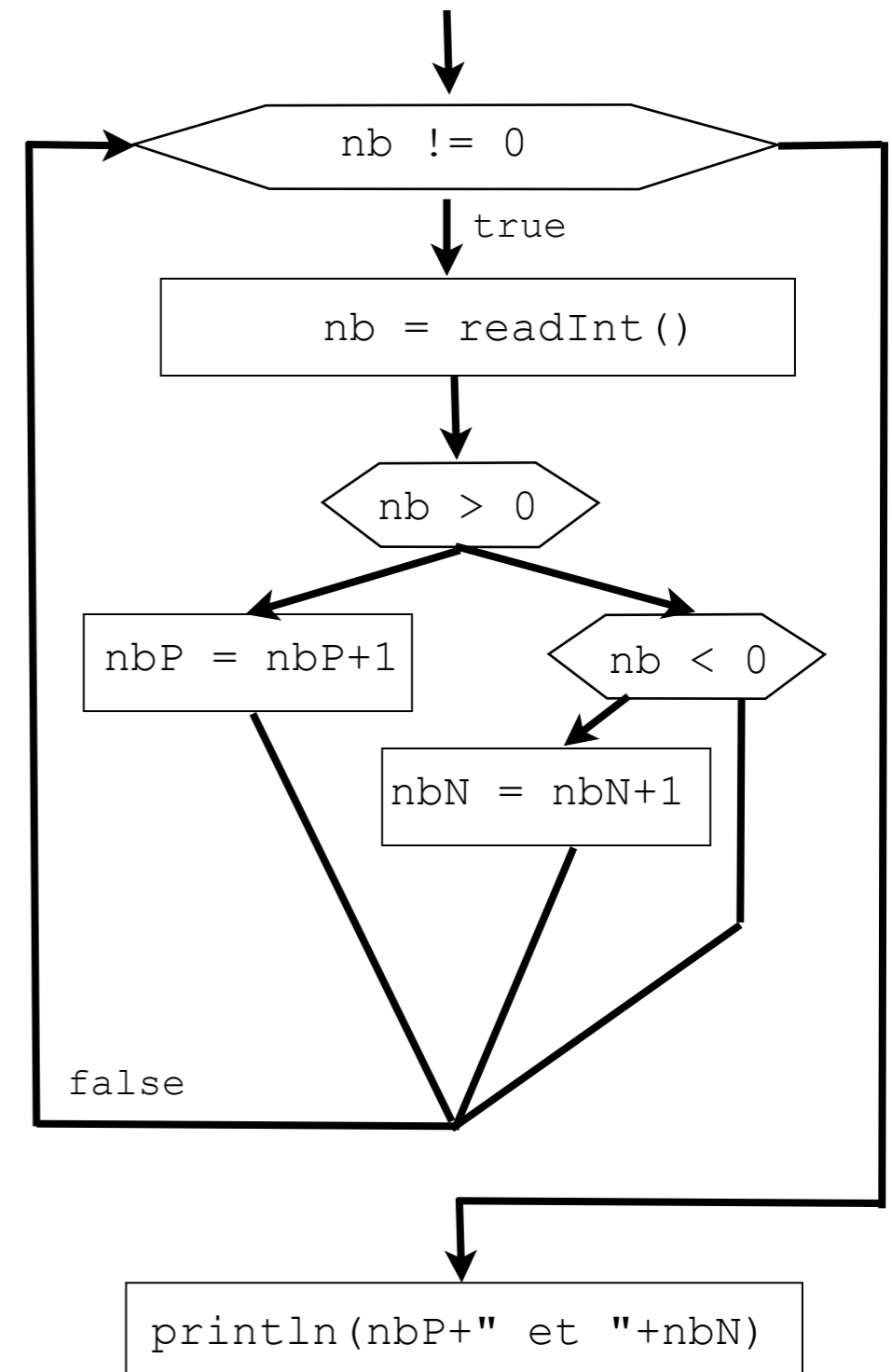
# Décompte de nombres

```
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);  
    }  
}
```



# Décompte de nombres

```
class CompterPositifNegatif extends Program {  
    void algorithm() {  
        int nbPositif = 0, nbNegatif = 0;  
        int nb = -1; // ou toute valeur != 0  
  
        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  $n! = 1 \times 2 \times \dots \times n$
- Exemple :  $5! = 5 * 4 * 3 * 2 * 1 = 120$
- Première version: `dowhile`
- Deuxième version: `for`

# Factorielle avec dowhile

```
class FactorielleDoWhile extends Program {  
    void algorithm() {  
        println("n ? ");  
        int n      = readInt();  
        int cumul  = 1;  
        int cpt    = n;  
        do {  
            cumul = cumul * cpt;  
            cpt = cpt - 1;  
        } while (cpt != 0);  
        println(n+"! = "+cumul);  
    }  
}
```

# Factorielle avec dowhile

```
...
int cumul = 1;
int cpt   = n;
if (n>1) {
    do {
        cumul = cumul * cpt;
        cpt = cpt - 1;
    } while (cpt != 0);
}
println(n+"! = " + cumul);
...
```

# Factorielle avec while

```
void algorithm() {  
    print("n ? ");  
    int n = readInt();  
    int cumul = 1;  
    int cpt = n;  
    while (cpt > 1) {  
        cumul = cumul * cpt;  
        cpt = cpt - 1;  
    }  
    println(n + "! = " + cumul);  
}
```

# Factorielle avec `for`

```
void algorithm() {  
    print("n ? ");  
    int n = readInt();  
    int cumul = 1;  
    for (int cpt = 2; cpt <= n; cpt = cpt+1) {  
        cumul = cumul * cpt;  
    }  
    println(n + "! = " + cumul);  
}
```

# Factorielle avec `for`

```
void algorithm() {  
    print("n ? ");  
    int n = readInt();  
    int cumul = 1;  
    for (int cpt = 2; cpt <= n; cpt = cpt+1) {  
        cumul = cumul * cpt;  
    }  
    println(n + "! = " + cumul);  
}
```

*Car  $0! = 1! = 1$*

*ce qui correspond à la valeur initiale de l'accumulateur :)*