

IFT 1015 - Boucles

Professeur:
Stefan Monnier

B. Kégl, S. Roy, F. Duranleau, S. Monnier
Département d'informatique et de recherche opérationnelle
Université de Montréal

hiver 2006

Au programme

- Boucles **en général**
- Boucles `while`, `do ... while` et `for`
- Indentation
- `Keyboard.error()` et `Keyboard.isEof()`
- Boucles **infinies**
- Les erreurs ± 1
- Boucles **imbriquées**

[Tasso:4], [Niño: 12.3, 12.6, 12.7]

Les boucles de style “recette”

“pour chacun des 8 oeufs, séparer le jaune du blanc”

au lieu de

“prendre le 1^{er} oeuf, séparer le jaune du blanc”

“prendre le 2^e oeuf, séparer le jaune du blanc”

...

“prendre le 8^e oeuf, séparer le jaune du blanc”

Les boucles de l'ordinateur

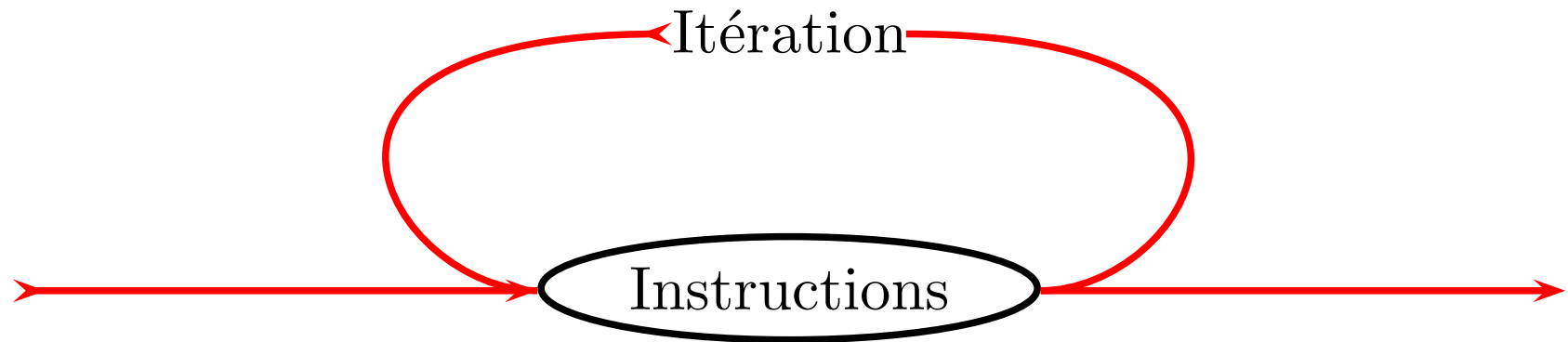
Calculer le prix de 100 billets d'avions:

- répéter le code du calcul 100 fois?
- et si l'utilisateur peut déterminer le nombre de billets lors de l'exécution?

Boucles (structures de répétition)

- moyens d'effectuer des répétitions

Principe fondamental d'une boucle

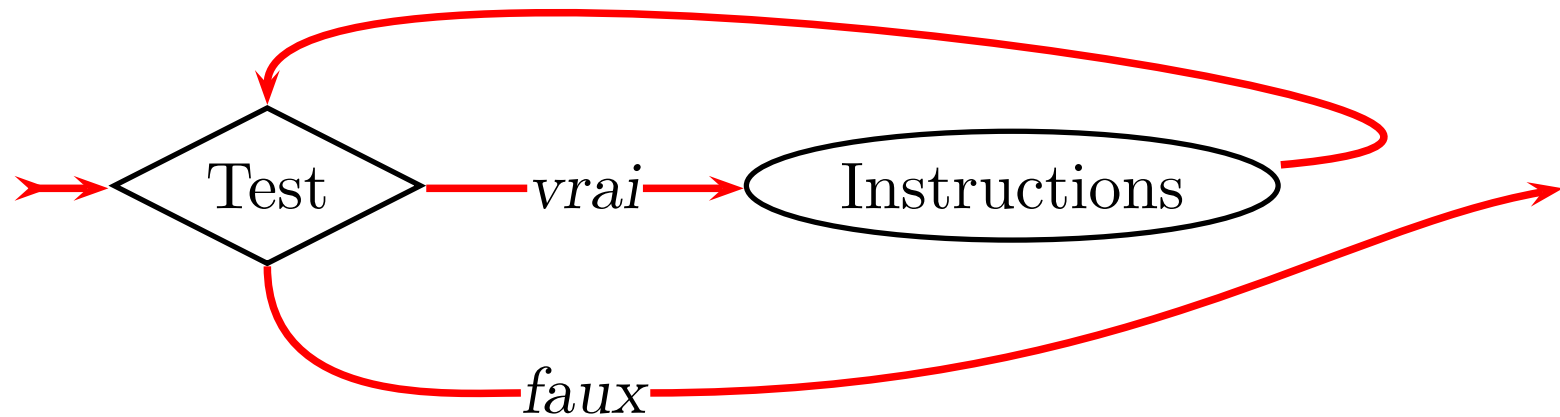


Itération: un processus dans lequel une **opération** est effectuée **plus d'une fois**

Combien de fois?

- il faut éventuellement **arrêter l'itération**
- quand une **condition** devient fausse: boucle **while** (tant que)
- ***n* fois**: boucle **for** (pour)

La boucle `while`



Si le test est

- *vrai* \Rightarrow on **entre dans la boucle** (une première fois ou à nouveau)
- *faux* \Rightarrow on **contourne la boucle** et l'exécution poursuit son cours

Après l'exécution du **corps**, on **retourne** à l'évaluation du **test**

Exemple: Machine à café

1. Poser la question: "Désirez-vous du sucre?"
2. Attendre la réponse.
3. WHILE la réponse est "OUI":
 4. Prendre un morceau de sucre.
 5. Déposer le morceau dans la tasse.
 6. Poser la question:
 7. "Désirez-vous un autre morceau de sucre?"
 8. Attendre la réponse.

Boucler à l'infini

Si l'évaluation du test est *toujours vrai*, on bouclera à l'infini

- erreur difficile à trouver

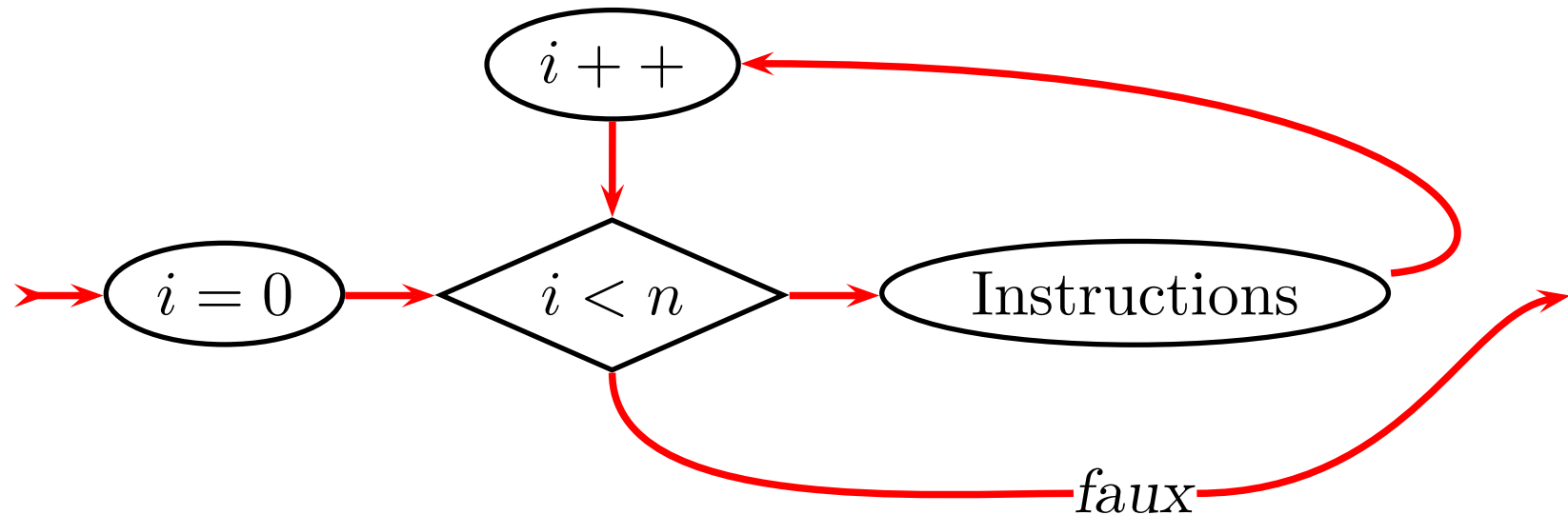
Pour éviter

- il doit y avoir une *dépendance* entre le *test* et le *corps* de la boucle
- dans l'exemple: le test dépend de la réponse de l'utilisateur

Exemple: Machine à café numérique

1. Poser la question:
 "Combien de morceaux de sucre désirez-vous?"
2. Attendre la réponse.
3. $n = 0$.
4. WHILE $n < \text{réponse}$:
5. Prendre un morceau de sucre.
6. Déposer le morceau dans la tasse.
7. $n = n + 1$.

Un schéma fréquent de boucles



Boucle `for`: spécialisation d'une boucle `while`

- but: exécuter les instructions n fois
- **compteur** $i = 0, 1, \dots, n - 1$: initialisé, incrémenté
- **variantes**: $i = \text{initVal}, i--, i \geq 0, i += 2, i *= 10$

Exemple `for` en pseudocode

1. Poser la question:

"Combien de morceaux de sucre désirez-vous

2. Attendre la réponse.

3. `FOR i = 1 TO réponse:`

4. Prendre un morceau de sucre.

5. Déposer le morceau dans la tasse.

- L'**initialisation** et le **test** de fin de boucle sont "**camouflés**" dans la spécification de l'intervalle `i = 1 TO réponse`
- L'**incrémentation** est **implicite**

Autre exemple

Afficher les nombres **pairs** entre 0 et n (inclusivement)

1. FOR $i = 0$ TO n STEP 2
2. Afficher i .

Équivalent à

1. $i = 0$
2. WHILE $i \leq n$:
3. Afficher i .
4. $i += 2$

Afficher les nombres pairs entre 0 et n exclusivement pour n

1. FOR $i = 0$ TO $n-1$ STEP 2

2. Afficher i .

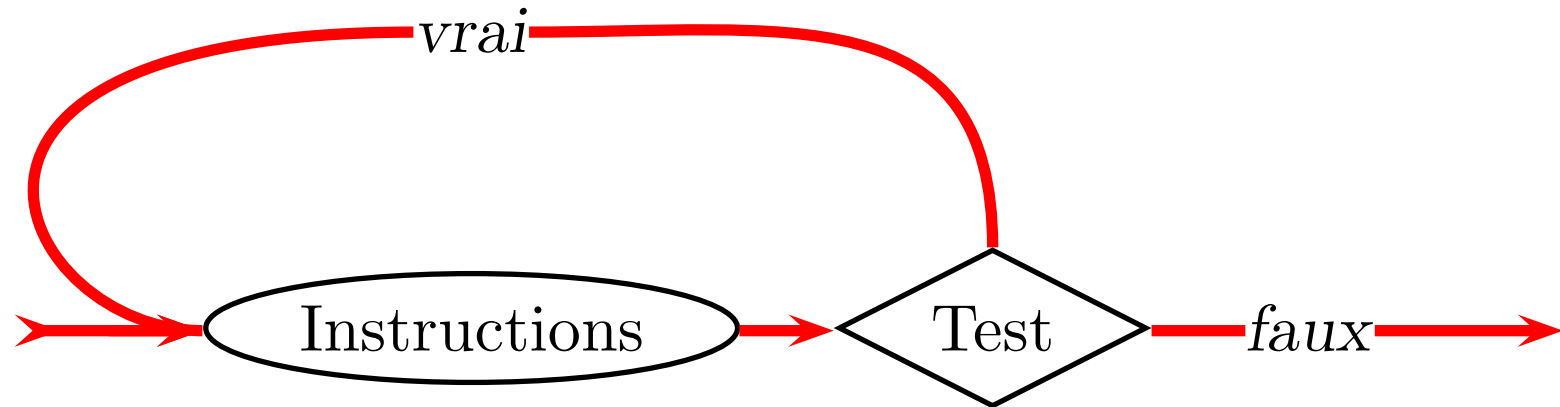
1. $i = 0$

2. WHILE $i \leq n-1$ ou WHILE $i < n$:

3. Afficher i .

4. $i += 2$

La boucle `do ... while`



Entrer dans la boucle **inconditionnellement**

Exemple: saisie au clavier avec **vérification**

1. DO
2. Demander l'entrée.
3. Attendre la réponse.
4. WHILE réponse est invalide

Et tout ceci, en **Java?**

La boucle `while`

Objectif: répéter un groupe d'instructions **tant qu'une condition est satisfaite**

```
while (<condition>)  
    <instruction>
```

- normalement `<instruction>` = **bloc d'instructions**
- comme pour `if`: toujours utiliser un bloc d'instructions

La boucle `do ... while`

Objectif: répéter un groupe d'instructions tant qu'une condition est satisfaite, mais **au moins une fois**

```
do
    <instruction>
while (<condition>)
```

- utilisé **rarement**
- normalement `<instruction>` = **bloc d'instructions**
- comme pour `while`: toujours utiliser un bloc d'instructions

Schéma fréquent

```
int i = start;
while (i < end) {
    <séquence d'instructions>
    i++;
}
```

Même programme avec `for`

```
for (int i = start; i < end; i++) {
    <séquence d'instructions>
}
```

La boucle `for`

Objectif: répéter un groupe d'instructions n fois (où n est pré-spécifié)

```
for (<init>; <condition>; <pas>)  
    <instruction>
```

- normalement `<instruction>` = bloc d'instructions
- comme pour `while`: toujours utiliser un bloc d'instructions

La boucle `for`

En principe: `for` peut contenir n'importe quoi Le plus souvent:

- `<init>` initialise un compteur:

```
int i = 0
```

- `<pas>` in/décrémente le compteur:

```
i++ ou i -= 5
```

- `<condition>` vérifie si le compteur est plus grand/plus petit qu'un seuil:

```
i <= 10 ou i > 0
```

Attention!

```
int n = 0;  
while (n < 10);  
    System.out.println(n);  
    n = n + 1;
```

Attention!

```
int n = 0;  
while (n < 10);  
    System.out.println(n);  
    n = n + 1;
```

Boucle infinie!!

```
int n = 0;
while (n < 10) {
    System.out.println(n);
    n = n + 1;
}
```

Toujours utiliser des **blocs**

Toujours faire attention de bien indenter **en fonction des blocs**

Bon exemple

```
int nb;
boolean valide = false;
do {
    System.out.print(
        "Entrez un nombre entre 1 et 100: ");
    nb = Keyboard.readInt();
    if ((nb < 1) || (nb > 100)) {
        System.err.println("Entrée invalide.");
    } else {
        valide = true;
    }
} while (!valide);
```

Mauvais exemple

```
int nb;
    boolean valide = false;
do {
    System.out.print(
        "Entrez un nombre entre 1 et 100: ");
        nb = Keyboard.readInt();
        if ((nb < 1) || (nb > 100)) {
            System.err.println("Entrée invalide.");
        } else {
            valide = true;
        }
    } while (!valide);
```


Directives pour bien indenter

Toute instruction dans un même bloc doit débiter à la même colonne

Indentation d'au moins deux espaces (par rapport au bloc englobant)

```
englobant
```

```
{
```

```
    indenté
```

```
    . . .
```

```
}
```

```
. . .
```

Directives pour bien indenter

Même nombre d'espaces pour **tous les blocs**

```
englobant1
{
    indenté1
    ...
    englobant2
    {
        indenté2
        ...
    }
}
```

Directives pour bien indenter

Instruction **coupée**: la suite doit être indentée.

De préférence, couper avant un opérateur plutôt qu'après.

```
grandTotal = total + fraisBase  
    + (int)((total + taxe) * TVQ + 0.5) + etc;
```

```
System.out.println("Le cercle de rayon "  
    + r + " a un perimètre de "  
    + p);
```

Enfin, plus d'**exemples**

Exemple 1

Saisie au clavier avec **vérification**

- lire un entier au clavier
- si l'utilisateur entre **autre chose** qu'un entier, recommencer la lecture
- **tant que** ce **n'est pas** un entier **valide**

Quel type de boucle?

- on doit lire **au moins une fois** \Rightarrow `do ... while`

Exemple 1

L'algorithme (pseudocode):

1. DO
2. Lire n.
3. WHILE n n'est pas un entier.

À propos de `Keyboard`

`Keyboard.error()` : expression booléenne

- `true` si la dernière lecture faite avec `Keyboard.read...()` a généré une erreur

Exemples

- `Keyboard.readDouble()`, l'utilisateur a tapé une lettre
- `Keyboard.readInt()`, l'utilisateur a tapé un nombre réel

Exemple 1: premier essai

```
int n;  
do {  
    n = Keyboard.readInt();  
} while (Keyboard.error());
```


Exemple 1: premier essai

```
int n;  
do {  
    n = Keyboard.readInt();  
} while (Keyboard.error());
```

Messages?

Exemple 1: 2^e essai

```
int n;  
do {  
    System.out.print("Entrez un entier: ");  
    n = Keyboard.readInt();  
} while (Keyboard.error());
```

Exemple 1: 2^e essai

```
int n;  
do {  
    System.out.print("Entrez un entier: ");  
    n = Keyboard.readInt();  
} while (Keyboard.error());
```

Message d'[erreur](#)?

Exemple 1: 3^e essai

```
int n;  
do {  
    System.out.print("Entrez un entier: ");  
    n = Keyboard.readInt();  
    if (Keyboard.error()) {  
        System.err.println("Entrée invalide.");  
    }  
} while (Keyboard.error());
```

Exemple 1: 3^e essai

```
int n;  
do {  
    System.out.print("Entrez un entier: ");  
    n = Keyboard.readInt();  
    if (Keyboard.error()) {  
        System.err.println("Entrée invalide.");  
    }  
} while (Keyboard.error());
```

redondance et cohérence?

Exemple 1: solution finale

```
int n;           // le nombre lu.
boolean invalide; // indique une erreur de lecture.
do {
    System.out.print("Entrez un entier: ");
    n = Keyboard.readInt();

    // un premier test pour signaler l'erreur:
    invalide = Keyboard.error(); // appelée 1 seule fois!
    if (invalide) {
        System.err.println("Entrée invalide.");
    }
} while (invalide);
```

Exemple 2

Somme d'une séquence

- lire une **séquence de nombre** au clavier
- tant qu'une **fin d'entrée** n'est pas lue
- calculer leur **somme**

Quel type de boucle?

- fin d'entrée peut se produire avant le premier nombre \Rightarrow `while`

Comment **signaler la fin** des commandes ou entrées

- **C-d**, si on lit au clavier
- **fin de fichier**, si on lit d'un fichier

Comment détecter?

- `Keyboard.isEof()` : expression booléenne
- `true` si la fin d'entrée a été lue lors de la **dernière**
`Keyboard.read...()`

Exemple 2: algorithme

Algorithme (pseudocode):

```
1. somme = 0
2. Lire valeur.
3. WHILE la fin d'entrée n'est pas lu:
4.     somme += valeur
5.     Lire valeur.
```

ou

```
1. somme = 0
2. DO
3.     Lire valeur.
4.     IF la fin d'entrée n'est pas lu:
5.         somme += valeur
6. WHILE la fin d'entrée n'est pas lu.
```

Exemple 2: code

```
int somme = 0; // initialisation de la somme
int valeur = Keyboard.readInt(); // valeur courante, 1ere lecture

// Tant que PAS à la fin d'entrée (d'où le '!').
while (!Keyboard.isEof()) {
    somme += valeur; // accumulation de la somme.
    valeur = Keyboard.readInt(); // lecture de la valeur suivante
}

// Affichage de la somme produite.
System.out.println("La somme est " + somme);
```

Affectation + arithmétique

$$n += v \equiv n = n + v$$

$$n -= v \equiv n = n - v$$

$$n *= v \equiv n = n * v$$

$$n /= v \equiv n = n / v$$

Incrémenter/décrémenter

$$n++ \simeq ++n \equiv n += 1 \equiv n = n + 1$$

$$n-- \simeq --n \equiv n -= 1 \equiv n = n - 1$$

Exemple 3: Intérêt

- étant donné le **solde initial**, le **solde souhaité** et le **taux d'intérêt**, **combien d'années** seront nécessaires pour atteindre le solde souhaité
- au lieu d'utiliser la formule, on **simule** le calcul

Algorithme (pseudocode):

1. `ans = 0;`
2. `WHILE` solde n'atteint pas à solde souhaité
3. incrémenter ans
4. ajouter l'intérêt au solde

Exemple 3: solution

```
int years = 0;
double balance = 1000;
double targetBalance = 1500;
double rate = 5; // en %.
while (balance < targetBalance) {
    years++;
    double interest = balance * rate / 100;
    balance += interest;
}
System.out.println(years + " years are needed");
```

Les erreurs ± 1

```
int years = 1; // devrait être 0
double balance = 1000;
double targetBalance = 1500;
double rate = 5; // en %
while (balance < targetBalance) {
    years++;
    double interest = balance * rate / 100;
    balance += interest;
}
System.out.println(years + "
    years are needed to reach " + targetBalance);
```

Les erreurs ± 1

```
int years = 1; // devrait être 0
double balance = 1000;
double targetBalance = 1500;
double rate = 5; // en %
while (balance < targetBalance) {
    years++;
    double interest = balance * rate / 100;
    balance += interest;
}
System.out.println(years + "
    years are needed to reach " + targetBalance);
```

Tester dans les **cas simples** ou **extrêmes**

Boucles infinies

```
int years = 0;
double balance = 1000;
while (years < 20) {
    double interest = balance * rate / 100;
    balance += interest; // years++; manque
}
```

```
int years = 20;
double balance = 1000;
while (years > 0) {
    years++; // devrait être years--
    double interest = balance * rate / 100;
    balance += interest;
}
```


Boucles infinies

```
int years = 0;
double balance = 1000;
while (years < 20) {
    double interest = balance * rate / 100;
    balance += interest; // years++; manque
}
```

```
int years = 20;
double balance = 1000;
while (years > 0) {
    years++; // devrait être years--
    double interest = balance * rate / 100;
    balance += interest;
}
```

Assurer que la **condition** du `while` deviendra éventuellement **fausse**

Exemple boucles imbriquées: objectif

afficher

```
[ ]  
[ ] [ ]  
[ ] [ ] [ ]  
[ ] [ ] [ ] [ ]  
[ ] [ ] [ ] [ ] [ ]  
[ ] [ ] [ ] [ ] [ ] [ ]  
[ ] [ ] [ ] [ ] [ ] [ ] [ ]  
[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
```

Exemple boucles imbriquées: étapes

Étape 1

```
int maxWidth = 7;
for (int i = 0; i < maxWidth; i++) {
    <afficher [] (i+1) fois>;
    <afficher une fin de ligne>;
}
```

Étape 2

```
{
    for (int j = 0; j < i+1; j++)
        System.out.print("[]");
    System.out.println();
}
```

Exemple boucles imbriquées: solution

```
int maxWidth = 7;
for (int i = 0; i < maxWidth; i++) {
    for (int j = 0; j < i+1; j++)
        System.out.print("[]");
    System.out.println();
}
```

Boucles imbriquées

Les instructions dans une boucle sont arbitraires \Rightarrow elles peuvent être d'autres boucles

Décomposer les problèmes en sous-problèmes plus simples

Exercices:

- afficher un rectangle avec seulement ses bordures
- afficher un triangle rectangle isocèle pour chaque orientation
- afficher un cercle plein!

Exemple 4: problème

- même problème qu'au [Lab 2](#)
- générer un [nombre aléatoire](#) entre 1 et 100
- donner [cinq chances](#) à l'utilisateur pour [deviner](#)
- avec indice de [trop grand/trop petit](#)

Quel type de boucle?

- exemple d'une répétition [avec compteur](#) \Rightarrow boucle `for`

Exemple 4: algorithme

```
1. Générer un nombre aléatoire entre 1 et 100.
2. Afficher: "Devinez un nombre entre 1 et 100 en 5 essais."
3. FOR essai = 1 TO 5:
4.     Afficher: "Essai #"
5.     Afficher essai.
6.     Lire nombre.
7.     IF nombre < magique:
8.         Afficher: "Trop petit!"
9.     ELSE IF nombre > magique:
10.        Afficher: "Trop grand!"
11.    ELSE:
12.        Afficher: "Réussi!"
13.        Quitter.
14. Afficher: "Perdu!"
```

Exemple 4: code

```
int nombreADeviner = (int) (Math.random() * 100) + 1;
System.out.println(
    "Devinez un nombre entre 1 et 100 en 5 essais.");
for (int esNum = 1; esNum <= 5; esNum++) {
    System.out.print("Essai #" + esNum + ": ");
    int devine = Keyboard.readInt();
    if (devine != nombreADeviner)
        System.out.println("Trop " + (devine > nombreADeviner
            ? "grand!" : "petit!"));
    else {
        System.out.println("Réussi!");
        System.exit(0);
    }
}
System.out.println("Perdu!");
```


Exemple 4: variantes

- l'utilisateur peut spécifier le nombre d'essais
- vérification d'entrée

Variante avancée

- en moyenne, combien d'essais sont nécessaires pour trouver le nombre?