Structures de répétition

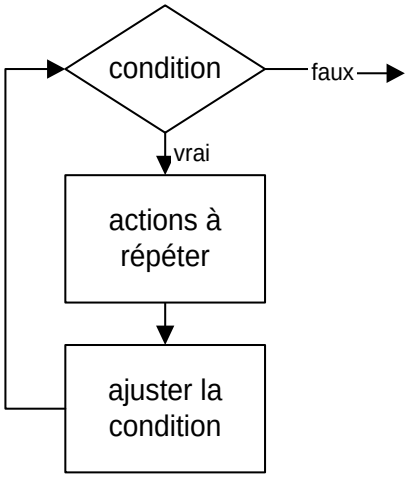
Les structures de répétition ou structures *itératives* permettent au programmeur de spécifier qu'une action sera répétée tant qu'une condition reste vérifiée.

1. La structure de répétition while… (tant que)

Algorithme :

```
tant que (condition)
    actions à répéter
    ajuster la condition
fin tant que
```

Ordinogramme :



Syntaxe en java :

```
while (condition)
{
    actions à répéter;
    ajuster la condition;
}
```

Fonctionnement :

a) Évaluer la condition
b) Si la condition est **vraie**
   1. exécuter les actions à répéter
   2. ajuster la condition
   3. retourner à a)
c) Si la condition est **fausse**, terminer la répétition.

Les actions à répéter sont exécutées 0 ou plusieurs fois. Si la condition est fausse la première fois qu'elle est évaluée, alors les actions ne seront jamais exécutées.

Remarques :

☞ Il faut faire bien attention de **ne pas** mettre `;` sur la ligne du while, car celui-ci serait considéré comme terminé et causerait une **boucle infinie** car la condition ne serait jamais modifiée.

```
while (nombre <= 10) ;
{
   System.out.println("Le nombre vaut " + nombre);
   nombre = nombre * 2;
}
```

Exemple 1 : on veut afficher les nombres 1 à 10 sur la console

Algorithme :

```
// initialiser les variables
nombre ← 1
// répéter 10 fois
tant que (nombre <= 10)
      afficher nombre
      nombre ← nombre + 1
fin tant que
```

La variable `nombre` *est la* **variable de contrôle** *de la boucle :*

1. *on l'initialise* ***avant*** *la boucle*
2. *on la vérifie dans la* ***condition*** *d'exécution de la boucle*
3. *on l'incrémente (avance)* ***dans*** *la boucle*

Programme en java :

```
public class Exemple1 {

    public static void main(String[] args) {

        // déclaration des variables
        int nombre = 1;

        // boucle de traitement
        while (nombre <= 10)
        {                                          // sépare chaque nombre
            System.out.print(nombre + "  ");       // par deux espaces
            nombre++;
        } // fin de la boucle
    }
}
```

Déroulement :

itération	traitement	nombre
0		1
1	afficher 1	2
2	afficher 2	3
3	afficher 3	4
4	afficher 4	5
5	afficher 5	6
6	afficher 6	7
7	afficher 7	8
8	afficher 8	9
9	afficher 9	10
10	afficher 10	11

Résultats obtenus sur la console :

```
1  2  3  4  5  6  7  8  9  10
```

Exemple 2 : on veut calculer et afficher la somme de 5 nombres entiers lus au clavier

Algorithme :

```
// initialiser les variables
somme ← 0
compteur ← 1
// répéter NB_FOIS fois
tant que (compteur <= NB_FOIS)
    lire nombre
    somme ← somme + nombre
    compteur ← compteur + 1
fin tant que
afficher somme
```

Données en entrée	Données en sortie
nombre	**somme**

Intermédiaires	Constantes
compteur	**NB_FOIS = 5**

L'utilisation de la constante **NB_FOIS** *permet de modifier facilement le nombre de répétitions de la boucle.*

Programme en java :

```
import javax.swing.*;
public class Exemple2 {

    public static void main(String[] args) {

        // déclaration des variables et des constantes
        final int NB_FOIS = 5;

        int nombre,
            somme = 0;              // somme des nombres
        int compteur = 1;           // variable de contrôle de la boucle

        // boucle de traitement des données
        while (compteur <= NB_FOIS)
        {
            // saisie d'un nombre
            nombre = Integer.parseInt
                      (JOptionPane.showInputDialog("Entrez un nombre:"));

            somme = somme + nombre;  // additionner le nombre
            compteur++;              // avancer le compteur
        } // fin de la boucle

        // affichage de la somme
        JOptionPane.showMessageDialog(null,
                    "La somme des nombres est " + somme);
        System.exit(0);
    }
}
```

Déroulement :

itération	nombre	somme	compteur
0		0	1
1	23	23	2
2	14	37	3
3	30	67	4
4	-4	63	5
5	9	**72**	6

La somme affichée sera de **72**

Exemple 3 : on veut calculer et afficher la somme de plusieurs nombres entiers lus au clavier

On ne connaît pas le nombre de nombres à traiter. On convient donc d'une valeur **sentinelle**, c'est-à-dire une valeur que l'usager doit entrer pour indiquer la fin des nombres.
Comme valeur sentinelle, on choisit habituellement une valeur en dehors des valeurs normales pour un nombre, ici **-1**.

Algorithme :

```
// initialiser les variables
somme ← 0
// lire le premier nombre
lire nombre
// répéter tant que pas lu la sentinelle
tant que (nombre != SENTINELLE)
    somme ← somme + nombre
    // lire le nombre suivant
    lire nombre
fin tant que
afficher somme
```

Données en entrée	Données en sortie
nombre	**somme**

Intermédiaires	Constantes
	SENTINELLE = -1

L'utilisation de la constante **SENTINELLE** *permet de repérer et modifier facilement la valeur sentinelle.*

Programme en java :

```
import javax.swing.*;
public class Exemple3 {

    public static void main(String[] args) {
        // déclaration des variables et des constantes
        final int SENTINELLE = -1;
        int nombre,
            somme = 0;          // total des nombres

        // lecture du premier nombre
        nombre = Integer.parseInt(JOptionPane.showInputDialog
                    ("Entrez un nombre ou " + SENTINELLE + " pour quitter:"));
        while (nombre != SENTINELLE)
        {
            somme += nombre;    // additionner le nombre

            // lecture du nombre suivant
            nombre = Integer.parseInt(JOptionPane.showInputDialog
                        ("Entrez un nombre ou " + SENTINELLE +
                         " pour quitter:"));
        } // fin de la boucle

        // affichage de la somme
        JOptionPane.showMessageDialog(null, "La somme est " + somme);
        System.exit(0);
    }
}
```

Déroulement :

itération	somme	nombre
0	0	5
1	5	12
2	17	8
3	25	-3
4	22	19
5	**41**	-1

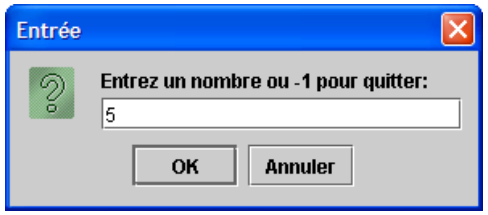
La somme affichée sera de **41**

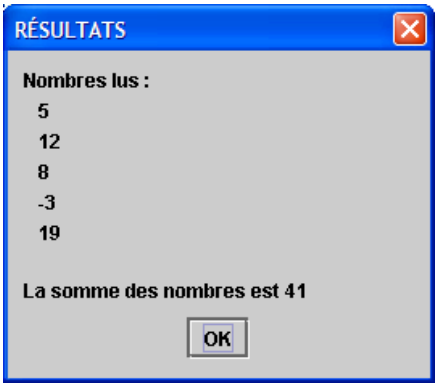
Exemple 3A : on veut calculer et afficher la somme de plusieurs nombres entiers lus au clavier

On reprend l'exemple précédent mais on veut, en plus, afficher chaque nombre lu. Pour obtenir une seule fenêtre de résultats, on cumule les affichages dans une variable String.

Programme en java :

```
import javax.swing.*;
public class Exemple3A {

    public static void main(String[] args) {
        // déclaration des variables et des constantes
        final int SENTINELLE = -1;

        String resultats = "Nombres lus :";   // ligne d'en-tête

        int nombre,
            somme = 0;          // total des nombres

        // lecture du premier nombre
        nombre = Integer.parseInt(JOptionPane.showInputDialog
                      ("Entrez un nombre ou " + SENTINELLE +
                       " pour quitter:"));
        while (nombre != SENTINELLE)
        {
            somme += nombre;   // additionner le nombre

            // envoi du nombre dans les résultats sur une nouvelle ligne
            resultats += "\n   " + nombre;

            // lecture du nombre suivant
            nombre = Integer.parseInt(JOptionPane.showInputDialog
                          ("Entrez un nombre ou " + SENTINELLE +
                           " pour quitter:"));
        } // fin de la boucle

        // affichage des résultats
        resultats += "\n\nLa somme des nombres est " + somme;
        JOptionPane.showMessageDialog(null, resultats, "RÉSULTATS",
            JOptionPane.PLAIN_MESSAGE);
        System.exit(0);
    }
}
```



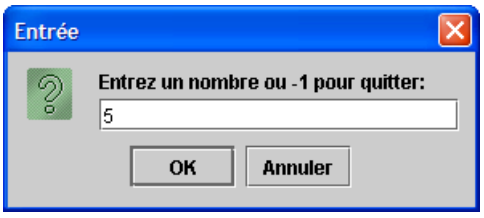


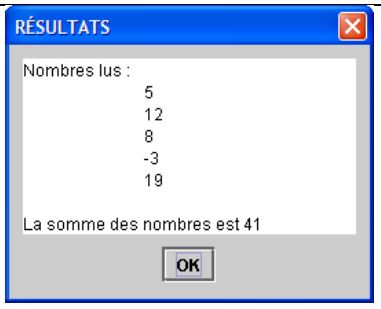
Exemple 3B : on veut calculer et afficher la somme de plusieurs nombres entiers lus au clavier

On reprend l'exemple précédent mais on veut pouvoir utiliser des **\t**. Pour obtenir une seule fenêtre de résultats et utiliser **\t**, on utilise un JTextArea.

Programme en java :

```
import javax.swing.*;
public class Exemple3B {

    public static void main(String[] args) {
        // déclaration des variables et des constantes
        final int SENTINELLE = -1;
        JTextArea sortie = new JTextArea();

        int nombre,
            somme = 0;          // total des nombres

        // ligne d'en-tête dans la sortie
        sortie.append("Nombres lus :");

        // lecture du premier nombre
        nombre = Integer.parseInt(JOptionPane.showInputDialog
                      ("Entrez un nombre ou " + SENTINELLE +
                       " pour quitter:"));
        while (nombre != SENTINELLE)
        {
            somme += nombre;    // additionner le nombre

            // envoi du nombre dans la sortie sur une nouvelle ligne
            sortie.append("\n\t" + nombre);

            // lecture du nombre suivant
            nombre = Integer.parseInt(JOptionPane.showInputDialog
                          ("Entrez un nombre ou " + SENTINELLE +
                           " pour quitter:"));
        } // fin de la boucle

        // affichage des résultats
        sortie.append("\n\nLa somme des nombres est " + somme);
        JOptionPane.showMessageDialog(null, sortie, "RÉSULTATS",
            JOptionPane.PLAIN_MESSAGE);
        System.exit(0);
    }
}
```





Exercice 1 : Faites l'algorithme qui permet de lire des températures tant que la valeur lue sera inférieure à 25. On doit ensuite afficher le nombre de températures lues.

Algorithme :

Données en entrée	Données en sortie

Intermédiaires	Constantes
	LIMITE = 25

Exercice 2 : Qu'affiche le programme suivant ?

```
public class Exercice2 {

    public static void main(String[] args) {
        int nb = 0,
            valeur1 = 0,
            valeur2 = 0;

        while (nb < 6)
        {
            switch (nb)
            {
                case 0:
                case 4:  valeur1++;
                case 1:
                case 5:  valeur2++;
                         break;
                default: System.out.println("Bonjour\n");
            }
            nb++;
        }
        System.out.println(valeur1 + "  " + valeur2);
    }
}
```

Résultats affichés sur la console :

Exercice 2 : Qu'affiche le programme suivant ?

```
public class Exercice3 {

    public static void main(String[] args) {
        int nb = 0,
            som = 0;

        while (nb < 20)
        {
            if (nb % 5 == 0)
            {
                som += nb;
                System.out.print(som + "  ");
            }
            nb++;
        }
        System.out.println("\nSomme = " + som);
    }
}
```

Résultats affichés sur la console :

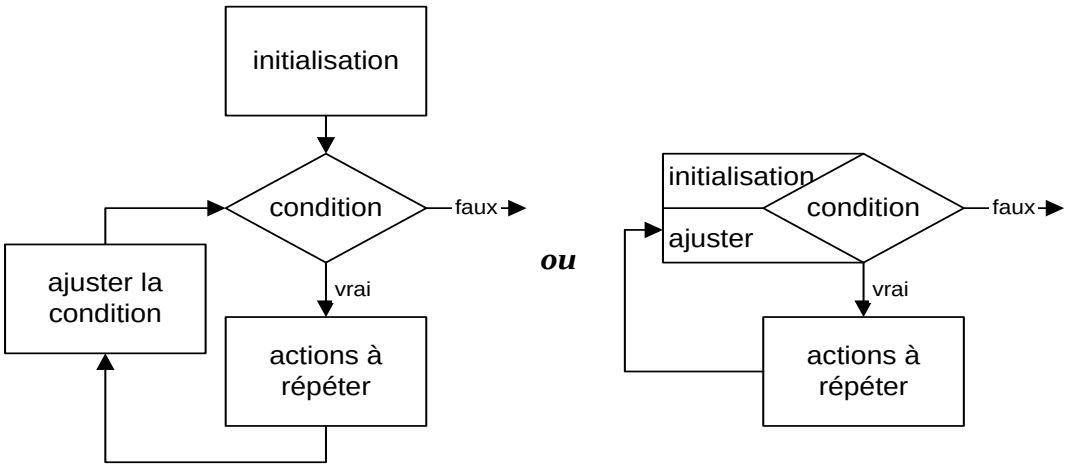
Exercice 3 : Qu'affiche le programme suivant ?

```
public class Exercice4 {

    public static void main(String[] args) {
        int nb1, nb2, nb3, som;
        nb1 = nb2 = som = 0;

        while (nb1 <= 4)
        {
            while (nb2 <= nb1 - 1)
            {
                nb3 = nb1 + nb2 - 1;
                if (nb3 % 2 == 0)
                    som = som + nb3;
                else
                    if (nb3 % 3 == 0)
                       som = som + nb3 - 2;
                System.out.print(som + "  ");
                nb2 = nb2 + 1;
            }
            nb1 = nb1 + 1;
        }
        System.out.println("\nSomme = " + som);
    }
}
```

Résultats affichés sur la console :

2. La structure de répétition for

Ordinogramme :



Syntaxe en java :

```
for (initialisation; condition; ajuster la condition)
{
    actions à répéter
}
```

Fonctionnement :

a) Faire l'initialisation
b) Évaluer la condition
c) Si la condition est **vraie**
   1. exécuter les actions à répéter
   2. ajuster la condition
   3. retourner à b)
d) Si la condition est **fausse**, terminer la répétition.

Les actions à répéter sont exécutées 0 ou plusieurs fois. Si la condition est fausse la première fois qu'elle est évaluée, alors les actions ne seront jamais exécutées.

Une structure for est équivalente à une structure while écrite de façon condensée et étant habituellement contrôlée par une variable de contrôle.

Remarques :

☞ Il faut faire bien attention de **ne pas** mettre `;` sur la ligne du for, car le bloc d'actions à répéter serait considéré comme **vide**.

```
for (nombre = 1; nombre <= 10; nombre++) ;
{
   System.out.println("Le nombre vaut " + nombre);
}
```

Exemple 1 : on veut afficher les nombres 1 à 10

Algorithme :

```
// répéter 10 fois
pour (nombre ← 1; nombre <= 10; nombre ← nombre + 1)
    afficher nombre
fin pour
```

Programme en java :

```
public class Exemple1 {

    public static void main(String[] args) {

        // déclaration des variables
        int nombre;

        // boucle de traitement
        for (nombre = 1; nombre <= 10; nombre++)
        {                                          // sépare chaque nombre
            System.out.print(nombre + "  ");       // par deux espaces
        } // fin de la boucle
    }
}
```

L'instruction for est utile pour initialiser la variable de contrôle, la tester pour la valeur limite et l'incrémenter dans la même instruction.

Exercice 1 : Pour chaque for suivant, indiquez le nombre d'itérations (le nombre de fois que la boucle sera effectuée)

	nombre d'itérations
`for (compt = 0; compt <= 5; compt++)`	
`for (m = 5; m > 0; m--)`	
`for (nb = 10; nb < 10; nb++)`	
`for (i = 0; i <= 10; i += 2)`	
`for (k = 1; k <= 10; k--)`	
`for (total = 50; total < 100; total += 10)`	
`for (nb = 100; nb >= 40; nb -= 20)`	

Exercice 2 : Complétez l'instruction `for` nécessaire pour faire afficher les nombres impairs compris entre 1 et 10

```
int nombre;
for (_____________________________________________)
    System.out.print(nombre + "  ");
```

Exercice 3 : Pour chaque bloc d'instructions, indiquez ce qui sera affiché sur la console

	affichage à l'exécution

`int i;`
`for (i = 10; i >= 1; i -= 1)`
`{`
`  System.out.print(i + "  ");`
`}`	

`int j;`
`for (j = 7; j <= 42; j = j + 7)`
`{`
`  System.out.print(j + "  ");`
`}`	

`int j;`
`for (j = 10; j <= 0 ; j++)`
`{`
`  System.out.print(j + "  ");`
`}`	

`int j;`
`for (j = 10; j >= 0; j += 1)`
`  System.out.print(j + "  ");`	

`for (int i = 1; i <= 10; i++)`
`  if (i % 3 == 0)`
`  System.out.print(i + "  ");`	

`int som = 0;`
`for (int m = 5; m >= 1, m--)`
`  if (m == 4)`
`  som = som - m;`
`  else`
`  som = som + m;`
`System.out.print("somme = " + som);`	

`for (int k = 5; k < 9; k++)`
`{`
`  j = k * 2 - 1;`
`  System.out.print(j + "  ");`
`}`	

Portée de la variable de contrôle :

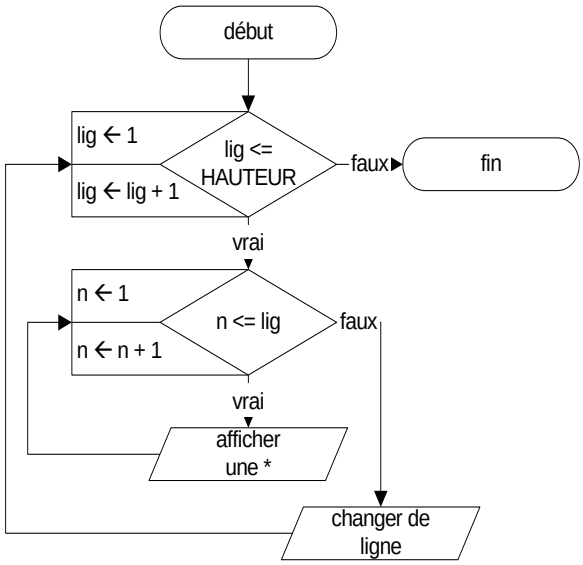
- ☞ Si la variable de contrôle d'un for est déclarée **dans** le for, comme ci-dessous, sa valeur n'est connue que dans l'instruction for et le bloc d'instructions que le for répète.

```
for (int i = 0; i < 10; i++)
{
    System.out.println("La valeur de i est " + i);
    somme += i;
}

// ce qui suit cause une erreur de compilation car i est inconnu
System.out.println("Après la boucle, i vaut " + i);
```

Exemple de boucle dans une boucle : On désire afficher un triangle sur la console

```
*
**
***
****
*****
```

Ordinogramme :



Programme en java :

```
public class Triangle {

    public static void main(String[] args) {

        // déclaration de la constante
        final int HAUTEUR = 5;

        // boucle extérieure pour faire HAUTEUR lignes
        for (int lig = 1; lig <= HAUTEUR; lig++)
        {
            // boucle intérieure pour faire lig "*" dans la ligne
            for (int n = 1; n <= lig; n++)
            {
                System.out.print("*");
            } // fin de la boucle intérieure

            System.out.println();    // pour changer de ligne
        } // fin de la boucle extérieure

    }
}
```

Exercice 4 : Complétez le programme qui permet d'afficher le triangle suivant sur la console

```
    *
   **
  ***
 ****
*****
```

Programme en java :

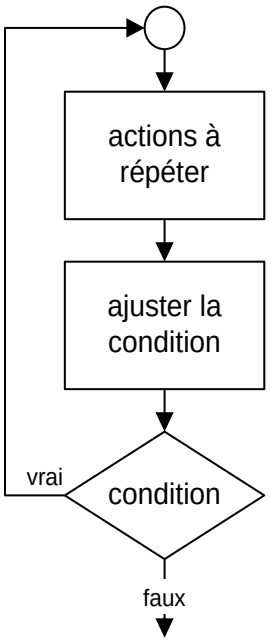
```
public class Triangle2 {

    public static void main(String[] args) {

        // déclaration de la constante
        final int HAUTEUR = 5;

        // boucle extérieure pour faire HAUTEUR lignes
        for (int lig = 1; lig <= HAUTEUR; lig++)
        {









            System.out.println();    // pour changer de ligne
        } // fin de la boucle extérieure

    }
}
```

3. La structure de répétition do...while (faire...tant que)

Algorithme :

faire
actions à répéter
ajuster la condition
tant que (*condition*)

Ordinogramme :



Syntaxe en java :

```
do
{
    actions à répéter;
    ajuster la condition;
} while (condition);
```

Fonctionnement :

a) Exécuter les actions à répéter
b) Ajuster la condition
c) Évaluer la condition
d) Si la condition est **vraie**, retourner à a)
e) Si la condition est **fausse**, terminer la répétition.

Les actions à répéter sont exécutées au moins une fois. Il s'agit de la différence majeure avec les structures while et for.

Exemple en java :

```
int somme = 1;
do
{
    somme = somme * 2;
} while (somme < 8);
```

Exemple 1 : on veut effectuer un traitement tant que l'usager n'a pas répondu 'N' à la question "Voulez-vous recommencer ?"

Comme on va effectuer un premier traitement avant de poser la question, on va utiliser une structure do...while.

Algorithme :

faire
    traitement quelconque
    poser question "Voulez-vous recommencer?" et lire réponse
tant que (réponse différente de 'N')

Programme en java :

```
public class Exemple1 {

    public static void main(String[] args) {

        // déclaration des variables
        char reponse;

        // boucle de traitement
        do
        {
            // traitement quelconque

            reponse = JOptionPane.showInputDialog(
                        "Voulez-vous recommencer (O/N) ?").charAt(0);
        } while (reponse != 'N');

        System.exit(0);
    }
}
```

Exemple 2 : on veut calculer et afficher la somme de plusieurs nombres entiers, en utilisant la valeur sentinelle 999.

Comparaison entre structures do...while et while :

```
// avec structure do...while

final int SENTINELLE = 999;
int nombre;
int somme = 0;

do
{
    // lire nombre
    nombre = Integer.parseInt(…);
    if (nombre != SENTINELLE)
    {
        somme = somme + nombre;
    }
} while (nombre != SENTINELLE);

System.out.println
    ("somme = " + somme);
}
```

```
// avec structure while

final int SENTINELLE = 999;
int nombre;
int somme = 0;

// lire premier nombre
nombre = Integer.parseInt(…);
while (nombre != SENTINELLE)
{
    somme = somme + nombre;
    // lire nombre
    nombre = Integer.parseInt(…);
}

System.out.println
    ("somme = " + somme);
}
```

La comparaison ci-dessus montre qu'on peut utiliser une ou l'autre de ces deux structures de répétition lorsqu'on veut exécuter une boucle arrêtée par la lecture d'une valeur sentinelle.

Dans le cas du do...while, il n'y a qu'une seule lecture. On doit donc évaluer si la note est différente de la valeur sentinelle avant de la traiter.

Dans le cas du while, on lit la première valeur **avant** la boucle et les suivantes **à la fin** de la boucle. Avec une valeur sentinelle, on utilise habituellement un while.

Exemple 3 : on veut lire au clavier un nombre entier compris entre 10 et 50 inclusivement. Si le nombre est invalide, on redemandera un autre nombre.

Algorithme :

```
faire
    lire nombre
tant que (nombre < 10 ou > 50)
afficher nombre, "est valide"
```

Programme en java :

```
import javax.swing.*;
public class Validation1 {

    public static void main(String[] args) {

        // déclaration des variables
        int nombre;

        // boucle de traitement
        do
        {
            nombre = Integer.parseInt(JOptionPane.showInputDialog(
                        "Entrez un nombre compris entre 10 et 50));

        } while (nombre < 10 || nombre > 50);

        // affichage du nombre valide
        JOptionPane.showMessageDialog(null, nombre + " est valide");
        System.exit(0);
    }
}
```

Si l'usager entre une valeur qui n'est pas comprise entre 10 et 50, la fenêtre de saisie réapparaît et il doit entrer un autre nombre.

Dans les pages suivantes, nous allons perfectionner cet exemple.

Exemple 3A : on veut lire au clavier un nombre entier compris entre 10 et 50 inclusivement. Si le nombre est invalide, on affichera un message d'erreur et on redemandera un autre nombre.

Algorithme :

```
faire
    lire nombre
    si (nombre < INF ou > SUP)
        afficher message d'erreur
    fin si
tant que (nombre < INF ou > SUP)
afficher nombre, "est valide"
```

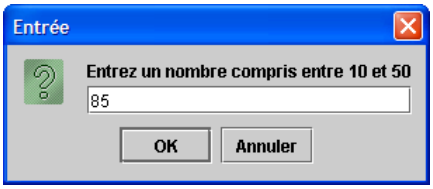
Constantes
INF = 10
SUP = 50

Programme en java :

```
import javax.swing.*;
public class Validation2 {

    public static void main(String[] args) {
        // déclaration des variables et des constantes
        final int INF = 10;
        final int SUP = 50;
        int nombre;

        // boucle de traitement
        do
        {
            nombre = Integer.parseInt(JOptionPane.showInputDialog(
                        "Entrez un nombre compris entre " +
                        INF + " et " + SUP));

            if (nombre < INF || nombre > SUP)
                JOptionPane.showMessageDialog(null,
                        "Le nombre est invalide", "ERREUR",
                        JOptionPane.ERROR_MESSAGE);

        } while (nombre < INF || nombre > SUP);

        // affichage du nombre valide
        JOptionPane.showMessageDialog(null, nombre + " est valide");
        System.exit(0);
    }
}
```

L'utilisation des constantes INF et SUP permet de modifier facilement les valeurs limites.

Si l'usager entre une valeur qui n'est pas comprise entre 10 et 50, un message d'erreur apparaît. Lorsque l'usager a lu le message et cliqué OK, la fenêtre de saisie réapparaît et il doit entrer un autre nombre.



Exemple 3B : on veut lire au clavier un nombre entier compris entre 10 et 50 inclusivement. Si le nombre est invalide, on affichera un message d'erreur dans la fenêtre de saisie du prochain nombre.

Algorithme :

```
message ← ""  // vide
faire
    afficher message et lire nombre
    message ← "Le nombre est invalide"
    valide ← (nombre >= INF et <= SUP)
tant que (pas valide)
afficher nombre, "est valide"
```

Données en entrée	Données en sortie
nombre	**message**

Intermédiaires	Constantes
valide	**INF = 10** **SUP = 50**

équivalent à

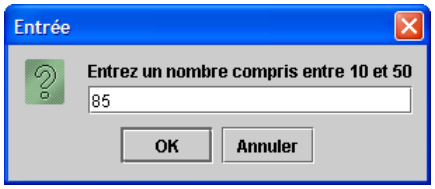
```
si (nombre >= INF et <= SUP)
   valide ← true
sinon
   valide ← false
fin si
```

Programme en java :

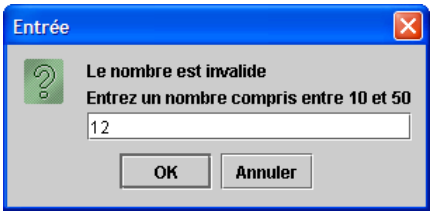
```
import javax.swing.*;
public class Validation3 {

    public static void main(String[] args) {

        // déclaration des variables et des constantes
        final int INF = 10;
        final int SUP = 50;
        int nombre;
        String message = "";
        boolean valide;

        // boucle de traitement
        do
        {
            nombre = Integer.parseInt(JOptionPane.showInputDialog(
                        message + "Entrez un nombre compris entre " +
                        INF + " et " + SUP));
            message = "Le nombre est invalide\n";
            valide = (nombre >= INF && nombre <= SUP);
        } while (!valide);

        // affichage du nombre valide
        JOptionPane.showMessageDialog(null, nombre + " est valide");
        System.exit(0);
    }
}
```

message *permet d'afficher le message d'erreur dans la fenêtre de saisie.*
valide *permet de simplifier la condition du while.*

Si l'usager entre une valeur qui n'est pas comprise entre 10 et 50, un message d'erreur apparaît **dans** la fenêtre de saisie où il doit entrer un autre nombre.



première fois



autres fois

Exercice 1 :

Complétez le programme offrant à l'usager le menu de la **figure 1**. Lorsque l'usager sélectionne :

- le **choix 1 ou 2**, on affiche la fenêtre de la **figure 2** (avec *Choix 1* s'il saisit 1)
- le **choix 3**, on affiche une fenêtre disant au revoir
- un **autre choix**, on affiche une fenêtre indiquant que son choix est invalide.

Le programme doit **toujours revenir au menu tant que le choix n'est pas 3**.

figure 1

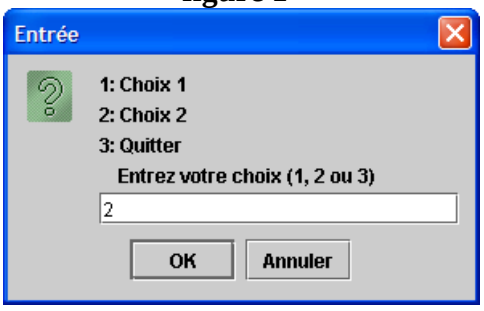
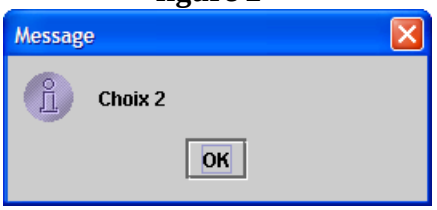


figure 2




```
/*
 * Menu.java
 */
import javax.swing.*;
public class Menu {

   public static void main(String[] args) {

        // déclaration des variables
        int choix;
```

```

```

```

```

```
            choix = Integer.parseInt(JOptionPane.showInputDialog
                        ("1: Choix 1\n2: Choix 2\n3: Quitter" +
                        "\n    Entrez votre choix (1, 2 ou 3)"));

            switch(choix)
            {
                case 1:
                case 2: JOptionPane.showMessageDialog(null,"Choix " + choix);
                        break;
                case 3: JOptionPane.showMessageDialog(null,"Au revoir");
                        break;
                default: JOptionPane.showMessageDialog(null,
                                "Choix invalide", "ERREUR",
                                JOptionPane.ERROR_MESSAGE);
            } //fin du switch
```

```

```

```

```
        System.exit(0);
    }
}
```
```

Exercice 2 : Faites l'algorithme qui permet de lire des nombres (un à la fois) tant que l'usager n'aura pas répondu 'N' à la question "Voulez-vous entrer un autre nombre (O/N) ?". On doit ensuite afficher la somme et le nombre de nombres lus.

Algorithme :

Données en entrée	Données en sortie

Intermédiaires	Constantes
	NON = 'N'

Exercice 3 : Qu'affiche le programme suivant ?

```
public class Exercice3 {

    public static void main(String[] args) {

        int nb = 0,
            som = 0;

        do
        {
            if (nb % 5 == 0)
            {
                som += nb;
                System.out.print(som + "  ");
            }
            nb++;
        } while (nb < 20);
        System.out.println("\nSomme = " + som);
    }
}
```

Résultats affichés sur la console :

Exercice 4 : Qu'affiche le programme suivant ?

```
public class Exercice4 {

    public static void main(String[] args) {
        int nb1 = 1,
            nb2,
            som = 0;

        while (nb1 < 3)
        {
            nb2 = 0;
            do
            {
                som = som + nb1 + nb2;
                som--;
                System.out.print(som + "  ");
                nb2++;
            } while (nb2 != nb1);
            nb1++;
        }
        System.out.println("\nSomme = " + som);
    }
}
```

Résultats affichés sur la console :

Exercice 5 : Qu'affiche le programme suivant ?

```
public class Exercice5 {

    public static void main(String[] args) {
        int nb;

        for (int i = 1; i <= 4; i++)
        {
            nb = i;
            do
            {
                System.out.print(nb + "  ");
                nb++;
            } while (nb <= 4);
            System.out.println();
        }
    }
}
```

Résultats affichés sur la console :

4. Sommaire des structures de répétition

Problématique	Structure	Recommandation
On veut lire et traiter *n* données	for : while : do…while :	Meilleur choix Si *n* vaut 0, on est cuit car on a déjà traité une donnée avant de vérifier !
On veut lire et traiter des données jusqu'à la lecture d'une valeur sentinelle	for : pas approprié while : do…while :	Meilleur choix On doit vérifier la donnée lue avant de la traiter
On veut lire et traiter des données jusqu'à ce que l'usager entre OUI en réponse à une question	for : pas approprié while : pas approprié do…while :	Meilleur choix

```
for (nb = 0; nb < n; nb++)
{
   lire une donnée
   traiter la donnée
}
```

```
nb = 0;
while (nb < n)
{
   lire une donnée
   traiter la donnée
   nb++;
}
```

```
nb = 0;
do
{
   lire une donnée
   traiter la donnée
   nb++;
} while (nb < n);
```

```
lire une donnée
while (donnée != SENTINELLE)
{
   traiter la donnée
   lire une autre donnée
}
```

```
do
{
   lire une donnée
   si (donnée != SENTINELLE)
      traiter la donnée
   fin si
} while (donnée != SENTINELLE);
```

```
do
{
   lire une donnée
   traiter la donnée
   afficher la question et
   lire la réponse
} while (réponse == OUI);
```

5. Traitements usuels avec des boucles

Problématique	Traitement	
accumulateur (somme)	avant la boucle :	`somme = 0`
	dans la boucle :	`somme += qqchose`
	après la boucle :	`afficher somme` ***ou*** s'en servir pour calculer autre chose
compteur (nombre de)	avant la boucle :	`compt = 0`
	dans la boucle :	`compt++`
	après la boucle :	`afficher compt` ***ou*** s'en servir pour calculer autre chose
moyenne (implique un accumulateur et un compteur)	avant la boucle :	`compt = 0` `somme = 0`
	dans la boucle :	`compt++` `somme += qqchose`
	après la boucle :	`moyenne = somme / compt`
plus petite donnée	avant la boucle :	`min = trop grosse valeur`
	dans la boucle :	`if (donnée < min)` `  min = donnée`
	après la boucle :	`afficher min`
plus grande donnée	avant la boucle :	`max = trop petite valeur`
	dans la boucle :	`if (donnée > max)` `  max = donnée`
	après la boucle :	`afficher max`

Exercice : Faites l'algorithme et complétez le programme qui permet de lire 10 notes (nombres réels), trouver et afficher la note la plus élevée, calculer et afficher la moyenne de la classe. De plus, on désire afficher le nombre d'élèves qui ont coulé (note moins de 60).

Algorithme :

```
// initialiser les variables



// répéter NB_NOTES fois
pour (nb = 0; nb < NB_NOTES; nb++)

    // trouver la note la plus élevée


    // vérifier si échec


fin pour

// calculer et afficher les statistiques
```

Données en entrée	Données en sortie

Intermédiaires	Constantes
	NB_NOTES = 10 **PASSAGE = 60**

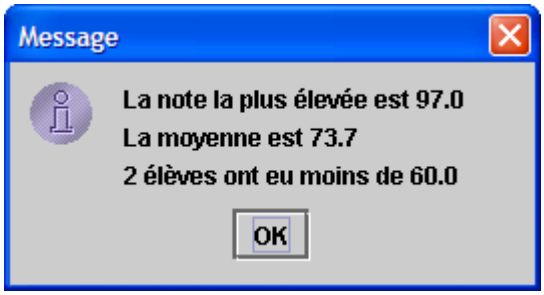
Programme en Java :

```
/*
 * Notes.java
 */
import java.text.*;
import javax.swing.*;
public class Notes {

    public static void main(String[] args) {

        // déclaration des variables et des constantes
        final int NB_NOTES = 10;
        final double PASSAGE = 60;
        DecimalFormat uneDecimale = new DecimalFormat("0.0");
```

```
        // répéter NB_NOTES fois




            // trouver la note la plus élevée


            // vérifier si échec


        } // fin de la boucle
        // calculer et afficher les statistiques



        System.exit(0);
    }
}
```

Résultats attendus :

Après avoir entré les notes suivantes : 80, 64, 58.2, 97, 60, 46, 75, 92.5, 68 et 95.9



FIN DE LA PARTIE 3