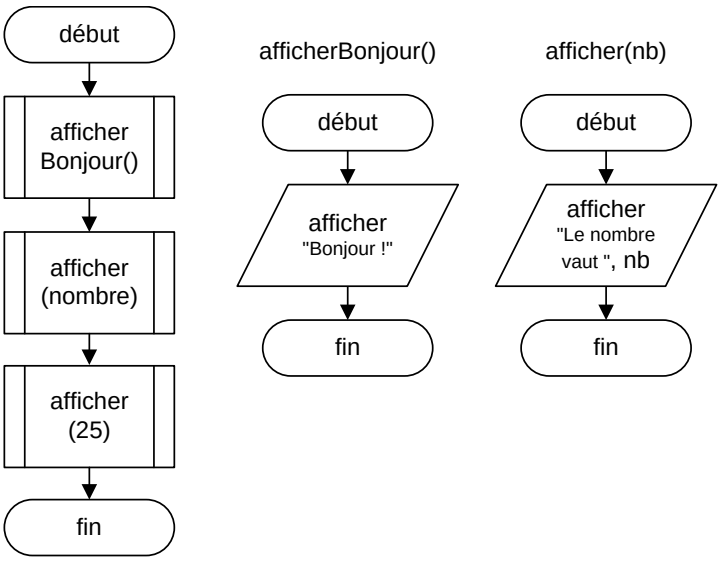
Méthodes

Les méthodes sont des blocs de programme contenant des déclarations et des instructions. Elles servent à dégager la logique du programme pour rendre celle-ci plus claire et plus facile à lire. Les méthodes permettent également de réutiliser du code au lieu de le répéter.

1. Structure d'un programme avec des méthodes

```
import javax.swing.*;
public class ExempleMéthode1
{
    public static void main(String[] args)
    {
        // déclaration des variables
        int nombre = 1;

        // appel aux méthodes
        afficherBonjour();     // demande d'afficher Bonjour
        afficher(nombre);      // demande d'afficher la variable nombre
        afficher(25);          // demande d'afficher la valeur 25

        System.exit(0);
    } // fin de la méthode main

    // méthode qui affiche Bonjour
    static void afficherBonjour()
    {
        JOptionPane.showMessageDialog(null, "Bonjour !");
    } // fin de la méthode afficherBonjour

    // méthode qui affiche un nombre entier reçu en paramètre
    static void afficher(int nb)
    {
        JOptionPane.showMessageDialog(null, "Le nombre vaut " + nb);
    } // fin de la méthode afficher

} // fin de la classe
```

Ordinogramme :



Algorithme en pseudo-code :

```
programme principal :
    nombre ← 1
    afficherBonjour()
    afficher(nombre)
    afficher(25)

méthode afficherBonjour() :
    afficher "Bonjour !"

méthode afficher(nb) :
    afficher "Le nombre vaut ", nb
```

2. Déclaration d'une méthode

Syntaxe :

```
// commentaires sur la méthode
static typeRetour nomMéthode(liste de paramètres)
{
    // déclaration des variables locales

    // instructions

} // fin de la méthode
```

signature de la méthode

corps de la méthode

`static` pour l'instant, toutes nos méthodes ont le modificateur **`static`**

`typeRetour` si la méthode ne retourne pas de résultat, **`typeRetour`** est **`void`**
si la méthode retourne un résultat, **`typeRetour`** indique le type du résultat

`nomMéthode` doit respecter les normes :
commence par une minuscule,
chaque mot suivant commence par une majuscule
contient souvent un verbe indiquant l'action
et un nom indiquant sur quoi porte l'action

`liste de paramètres` (dits *paramètres formels*)
chaque paramètre est composé d'un type et d'un nom,
on sépare les différents paramètres par des virgules

Exemples :

```
// méthode qui affiche Il fait beau aujourd'hui sur la console
// paramètres reçus : aucun
static void afficherBeauTemps()
{
    System.out.println("Il fait beau aujourd'hui");
} // fin de la méthode
```

méthode sans paramètre

```
// méthode qui affiche une phrase sur la console
// paramètres reçus : la phrase à afficher
static void afficherPhrase(String phrase)
{
    System.out.println(phrase);
} // fin de la méthode
```

méthode avec 1 paramètre

```
// méthode qui affiche 5 fois un caractère sur la console
// paramètres reçus : le caractère à afficher
static void afficher5Caracteres(char carac)
{
    for (int i = 1; i <= 5; i++)
        System.out.print(carac);

} // fin de la méthode
```

méthode avec 1 paramètre

```
// méthode qui affiche nb fois un caractère sur la console
// paramètres reçus : le caractère à afficher
//                    le nombre de fois désiré
static void afficherCaractere(char carac, int nb)
{
    for (int i = 1; i <= nb; i++)
        System.out.print(carac);

} // fin de la méthode
```

méthode avec 2 paramètres

```
// méthode qui retourne une chaîne de caractères formée de
// nb fois le caractère reçu en paramètre
// paramètres reçus : le caractère
//                    le nombre de fois désiré
static String repeterCaractere(char carac, int nb)
{
    String resultat = "";

    for (int i = 1; i <= nb; i++)
        resultat += carac;

    return resultat;
} // fin de la méthode
```

méthode avec 2 paramètres et valeur retournée

```
// méthode qui indique si le caractère est trouvé dans la chaîne ou non
// paramètres reçus : le caractère recherché
//                    la chaîne de caractères
static boolean estPresent(char carac, String chaine)
{
    boolean trouve;

    if (chaine.indexOf(carac) != -1)
        trouve = true;
    else
        trouve = false;

    return trouve;
} // fin de la méthode
```

méthode avec 2 paramètres et valeur retournée

ou

```
trouve = (chaine.indexOf(carac) != -1);
```

ou

```
return (chaine.indexOf(carac) != -1);
```

3. Appel d'une méthode

Syntaxe :

```
// si la méthode ne retourne aucune valeur
nomMéthode(liste de paramètres);

// si la méthode retourne une valeur
variable = nomMéthode(liste de paramètres);

// si la méthode retourne une valeur, on peut utiliser
// l'appel à la méthode dans n'importe quelle expression,
// de la même façon qu'on aurait utilisé une valeur ou une variable
```

`liste de paramètres` (dits *paramètres à l'appel*)

chaque paramètre (ou *argument*) est composé d'une expression (nom de variable, valeur ou expression),
on sépare les différents paramètres à l'appel par des virgules,
les paramètres doivent correspondre un pour un (en nombre et en type) avec les paramètres de la signature de la méthode (*paramètres formels*)

Passage des paramètres à une méthode :

Le passage *par valeur* :

- Transmet une **copie** de la valeur de l'argument à la méthode appelée
- Appliqué automatiquement pour des arguments de type primitif (**int**, **float**, **double**, **char**, **boolean**, etc)
- La méthode appelée **ne modifie pas** la valeur originale de l'argument

Le passage *par référence* :

- Transmet la référence (l'adresse) de l'argument à la méthode appelée
- Appliqué automatiquement pour des arguments de type non primitif (**String**, **tableau**, **objet**)
- La méthode appelée **peut modifier** la valeur originale de l'argument car elle a accès directement à l'argument (sauf pour String)

Exemples :

```
public class ExempleAppels {
    public static void main(String args[]) {
        afficherBeauTemps();

        afficherPhrase("Les oiseaux chantent");

        afficher5Caracteres('*');

        afficherCaractere('$', 10);

        String texte = repeterCaractere('*', 10);

        char cherche = 'a';
        String phrase = "L'hiver approche";
        if (estPresent(cherche, phrase))
            System.out.println(cherche + " est bien dans " + phrase);
        else
            System.out.println(cherche + " n'est pas dans " + phrase);
    } // fin de la méthode main
    // déclaration des méthodes
} // fin de la classe
```

Exercice 1 : Écrivez une méthode qui permet de déterminer si une année est bissextile

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

    public static void main(String args[]) {
        int annee;
        boolean bissextile;
        String message;

        annee = Integer.parseInt(JOptionPane.showInputDialog(
                              "Entrez une année (4 chiffres)");

        bissextile = ____________________________  // appel à la méthode

        if (bissextile)
            message = " est ";
        else
            message = " n'est pas ";

        JOptionPane.showMessageDialog(null,
                               annee + message + "bissextile");
        System.exit(0);
    } // fin de la méthode main

    // méthode qui indique si une année est bissextile ou non
    // pour être bissextile, l'année doit se diviser par 400 ou,
    // si ce n'est pas un siècle, l'année doit se diviser par 4
    static __________ estBissextile(_________________)
    {
        boolean bissextile;









    } // fin de la méthode estBissextile

} // fin de la classe
```

Exercice 2 : Écrivez une méthode qui permet de calculer le périmètre d'un rectangle

```
public class Perimetre {

    public static void main(String args[]) {
        int largeur = 5;
        int hauteur = 3;

        System.out.println("Le périmètre du rectangle est " +
            ____________________________________________________);

    } // fin de la méthode main

    // méthode qui calcule le périmètre d'un rectangle
    // paramètres reçus : largeur et hauteur du rectangle
    static __________ calculerPerimetre(_____________________________________)
    {









    } // fin de la méthode calculerPerimetre

} // fin de la classe
```

Exemple complet :

```
/* Modularite.java
 * Ce programme permet de saisir un nombre entier et d'afficher
 * la somme de 1 jusqu'à ce nombre.
 * Le processus recommence tant qu'on répond O (pour oui)
 * à la question "Voulez-vous recommencer (O/N)?"
 */

import javax.swing.*;
public class Modularite
{
    public static void main(String args[])
    {
        int nombreLu;
        char reponse;

        do
        {
            nombreLu = Integer.parseInt(JOptionPane.showInputDialog(
                        "Entrez un nombre entier"));
            afficherSomme(nombreLu);
            reponse = obtenirReponseValide();
        } while (reponse == 'O');

        System.exit(0);
    } // fin du main

    // calculer et afficher la somme de 1 au nombre reçu en paramètre
    static void afficherSomme(int nombre)
    {
        int somme = 0;


        for (int nb = 1; nb <= nombre; nb++)
            somme += nb;


        JOptionPane.showMessageDialog(null,
            "La somme de 1 à " + nombre + " est de " + somme);
    } // fin de la méthode afficherSomme

    // lire et valider un caractère (O ou N)
    // retourner le caractère valide
    static char obtenirReponseValide()
    {
        char carac;

        do
        {
            carac = JOptionPane.showInputDialog(
                    "Voulez-vous recommencer (O/N) ?").charAt(0);
            carac = Character.toUpperCase(carac);
        } while (carac != 'O' &&  carac != 'N');

        return carac;
    } // fin de la méthode obtenirReponseValide

} // fin de la classe
```

Ces variables sont **locales** à la méthode **`main`**, elles ne sont pas connues dans les autres méthodes.

La variable **`somme`** est **locale** à la méthode, elle cesse d'exister lorsque la méthode se termine.

La variable **`nb`** est **locale** au **`for`**, elle cesse d'exister après le **`for`**.

4. Méthodes de la classe Math

La classe Math (se trouve dans la librairie *java.lang* - donc pas besoin de l'importer) donne accès à quelques constantes ainsi qu'à une panoplie de fonctions mathématiques telles sinus, cosinus, racine carrée, etc.

Constantes	
Math.E	constante e
Math.PI	constante pi
Méthodes	
type Math.abs(*type*)	Retourne la valeur absolue du nombre, dans le même type, type peut être *double*, *float*, *int* ou *long*
double Math.ceil(*double*)	Retourne l'entier (converti en *double*) immédiatement supérieur ou égal au nombre
double Math.floor(*double*)	Retourne l'entier (converti en *double*) immédiatement inférieur ou égal au nombre
type Math.max(*type*, *type*)	Retourne le plus grand de deux nombres de même type, type peut être *double*, *float*, *int* ou *long*
type Math.min(*type*, *type*)	Retourne le plus petit de deux nombres de même type, type peut être *double*, *float*, *int* ou *long*
double Math.pow(*double*, *int*)	Retourne la puissance du nombre
double Math.random()	Retourne un nombre aléatoire dans la plage [0, 1[
long Math.round(*double*)	Retourne l'entier (converti en *long*) le plus près du nombre
int Math.round(*float*)	Retourne l'entier le plus près du nombre
double Math.sqrt(*double*)	Retourne la racine carrée du nombre
Exemples	
Math.ceil(6.789)	vaut 7.0
Math.floor(6.789)	vaut 6.0
Math.floor(Math.PI)	vaut 3.0
Math.round(6.789)	vaut 7
Math.abs(2 - 8)	vaut 6
Math.sqrt(9.0)	vaut 3.0
Math.min(2.35, 5.0)	vaut 2.35
Math.pow(9.0, 2)	vaut 81.0
Math.random() * 6 + 1	valeur *double* entre 1.0 et 6.9999999...
(int)(Math.random() * 6) + 1	valeur *int* entre 1 et 6
Math.pow(Math.ceil(4.25), 3)	vaut 125.0, soit 5.0 exposant 3

Exercice 3 : Qu'affiche le programme suivant ?

```
public class Exercice3
{
   public static void main (String args[])
   {
      int nombre = 10;
      plus1(nombre);

      System.out.println(nombre + " ");
      plus1(nombre + 15);
      System.out.println(nombre);
   }

   static void plus1(int z)
   {
      z++;
      System.out.print(z + " ");
   }
}
```

Résultats affichés sur la console :

Exercice 4 : Qu'affiche le programme suivant ?

```
public class Exercice4
{
   public static void main (String args[])
   {
      int nombre = 10;
      nombre = plus1(nombre);
      System.out.println(nombre);
   }

   static int plus1(int z)
   {
      z++;
      System.out.print(z + " ");
      return z;
   }
}
```

Résultats affichés sur la console :

Exercice 5 : Qu'affiche le programme suivant ?

```
public class Exercice5
{
   public static void main (String args[])
   {
      int a = 5,
          b = 6,
          c = 7;

      System.out.println("Avant l'appel de test");
      System.out.print("a = " + a + " b = " + b +
                       " c = " + c + "\n\n");

      c = test(a, b);

      System.out.println("Après l'appel de test");
      System.out.print("a = " + a + " b = " + b +
                       " c = " + c);
   }

   static int test(int x, int y)
   {
      int z;

      x += 10;
      y += 10;
      z = x + y;
      System.out.println("À l'intérieur de test");
      System.out.print("x = " + x + " y = " + y +
                       " z = " + z + "\n\n");
      return z;
   }
}
```

Résultats affichés sur la console :

Exercice 6 : Complétez le programme suivant qui appelle la méthode *afficherMultiple*. La méthode affiche sur la console tous les multiples de 3 compris entre 0 et un nombre entier limite, reçu en paramètre.

```
public class Exercice6 {

    public static void main(String args[])
    {
        // appel de la méthode pour faire afficher
        // les multiples de 3 compris entre 0 et 12
        _____________________________________________

    } // fin de la méthode main

    // méthode qui affiche les multiples de 3
    // compris entre 0 et limite
    static __________ afficherMultiple(_____________________________)
    {
        System.out.println("Multiples de 3 compris entre 0 et " + _______);









    } // fin de la méthode afficherMultiple

} // fin de la classe
```

Exercice 6B : Modifiez le programme précédent pour que la méthode reçoive également en paramètre le nombre dont on veut afficher les multiples.

```
public class Exercice6B {

    public static void main(String args[])
    {
        // appel de la méthode pour faire afficher
        // les multiples de 2 compris entre 0 et 8
        _____________________________________________

    } // fin de la méthode main

    // méthode qui affiche les multiples de multiple
    // compris entre 0 et limite
    static __________ afficherMultiple(_____________________________________)
    {
        System.out.println("Multiples de " + __________ +
                           " compris entre 0 et " + __________);









    } // fin de la méthode afficherMultiple

} // fin de la classe
```

Exercice 7 : Écrivez une méthode *estEntier* qui reçoit une chaîne de caractères et qui s'assure qu'il n'y a que des chiffres dans cette chaîne. Votre méthode devra retourner **true** ou **false**.

```
//
//
static __________ estEntier(_____________________________________)
{









} // fin de la méthode estEntier
```

Exercice 8 : Écrivez une méthode *retrancherTexte* qui reçoit deux chaînes de caractères en paramètres. Votre méthode devra chercher la seconde chaîne dans la première et en enlever la première occurrence. Elle retourne ensuite la chaîne transformée.

```
//
//
static __________ retrancherTexte(_____________________________________)
{









} // fin de la méthode retrancherTexte
```

Exercice 9 : Écrivez une méthode *rechercherPlusGrand* qui retourne le plus grand de trois nombres entiers reçus en paramètres.

```
//
//
static __________ rechercherPlusGrand(_____________________________________)
{









} // fin de la méthode rechercherPlusGrand
```

Exercice 10 : Écrivez une méthode *calculerConsommation* qui reçoit en paramètres une distance en kilomètres ainsi que le nombre de litres consommés (deux nombres réels). La méthode retourne la consommation obtenue en litres par 100 kilomètres. Par exemple, pour une distance de 592.5 kilomètres et une consommation de 38.75 litres, on aura une consommation de 6.45 litres par 100 kilomètres.

```
//
//
static __________ calculerConsommation(_____________________________________)
{



} // fin de la méthode calculerConsommation
```

Exercice 11 : Écrivez la méthode *obtenirNombreValide* qui doit lire et valider un nombre selon l'algorithme de la page 156. La méthode retourne ensuite le nombre valide.

```
import javax.swing.*;
public class Exercice10 {

    public static void main(String args[])
    {
        JOptionPane.showMessageDialog(null,
           "Le nombre obtenu est " +
obtenirNombreValide());
        System.exit(0);
    }

    static __________ obtenirNombreValide()
    {









    } // fin de la méthode obtenirNombreValide
}
```