MODULARISATION

VALEUR DE RETOUR D'UNE MÉTHODE

MÉTHODE - RETOURNER UN RÉSULTAT

Les méthodes que nous avons vues jusqu'à présent étaient plutôt « orientées action » : `afficherCeci`, `dessinerCela`…

Les paramètres permettent de les rendre plus génériques : **afficher un rectangle de taille `longueur x largeur`**…

Les paramètres permettent de *transmettre* quelque chose à la méthode, mais pas vraiment de recevoir un résultat en retour

On va voir comment écrire des méthodes qui **renvoient** quelque chose à l'appelant

Ces méthodes répondent à des questions :

- *Quelle est la racine carrée de 12.5 ?*
- *Quel est le nom du client qui correspond à la référence de cette facture ?*

EXEMPLE

- Chez l'appelant, on voudrait ça :

```
int racine = calculerRacine(9.0);  // affectation du résultat à la variable 'racine'
```

Mais pour l'instant, on ne sait définir de méthodes qui ne font que des actions, comme afficher :

```
public static void calculerRacine(double n) {
  double racine = ...;
  System.out.println(racine);
}
```

On voudrait **renvoyer** la valeur calculée à l'appelant, pour qu'il puisse utiliser le résultat, et non pas simplement l'afficher dans notre méthode

TYPE DE RETOUR

Dans une définition de méthode, le mot juste avant le nom de la méthode définit le **type de retour** de la méthode

Jusqu'à présent, c'était toujours `void` (vide) : nos méthodes ne retournaient rien

On va remplacer ce mot-clé dans l'entête de la méthode par le type de la valeur qu'on veut retourner. La racine carré d'un double est aussi un double :

L'instruction `return` est utilisée pour renvoyer la valeur à l'appelant

```
public static double calculerRacine(double n) {
  double racine = ...;
  return racine;
}
```

L'INSTRUCTION RETURN

```
return <expression>;
```

Quand Java rencontre l'instruction `return`, il **évalue l'expression** puis la **renvoie à l'appelant**

Conséquence : `return` **termine immédiatement la méthode** (retour à l'appelant)

`return` est **obligatoire** dans toute méthode dont le type de retour n'est pas `void`

Tous les types sont possibles comme type de retour d'une méthode

EXEMPLE

Écrivons une méthode `sommeDesEntiersJusque(int n)` qui calcule et retourne la somme des n premiers entiers : `1 + 2 + 3 + … + n`

```java
public static int sommeDesEntiersJusque(int n) {
  int somme = 0;
  for (int i = 1; i <= n; i++) {
    somme += i;
  }
  return somme;
}
```

```java
int somme = sommeDesEntiersJusque(100);
System.out.println("Somme : " + somme);


______
SORTIE

5050
```

ERREUR COMMUNE - IGNORER LA VALEUR DE RETOUR

Si vous utilisez une méthode avec un type de retour, en général vous voudrez récupérer cette valeur (chez l'appelant)

Il est fréquent (et même souhaitable) que ces méthodes ne fassent rien d'autre que calculer la valeur de retour

Donc, si vous ne la récupérez pas, le traitement est perdu :

```
sommeDesEntiersJusque(1000); // résultat dans la nature
Math.round(2.6);             // résultat dans la nature
```

RÉCUPÉRER ET UTILISER LE RÉSULTAT

Vous pouvez utiliser un appel de méthode non-`void` dans n'importe quelle expression : à droite d'une affectation, au milieu d'une expression, comme argument dans un autre appel de méthode, etc.

```
int resultat = sommeDesEntiersJusque(1000); // OK : résultat pas perdu

double unFlottant = 2.5;
// Expression impliquant des appels de méthodes
double nb = Math.ceil(unFlottant) + Math.floor(unFlottant);

// Expression impliquant des appels de méthodes imbriqués
double racine = Math.sqrt(sommeDesEntiersJusque(Math.round(98.76)));

// Utilisation dans un println : toujours pareil,
// le résultat est utilisé comme valeur pour l'opérande.
// L'expression est totalement évaluée avant d'être affichée.
System.out.println("Un nombre aléatoire entre 0 et 1 : " + Math.random());
```