R1.01 Initiation au développement

Principe (par l'exemple)

1. Factorisation

```
// séquence sans fonction
//
écrire("paragraphe 1")
sauter une ligne          |
sauter une ligne          | {répétition à chaque}
sauter une ligne          |  {nouveau paragraphe}
sauter trois espaces      |
écrire("paragraphe 2")
```

⇒

```
// fonction nouveauParagraphe()
// saute trois lignes et indente
// de trois espaces sur la quatrième
//
Fonction nouveauParagraphe()
Début
        sauter une ligne
        sauter une ligne
        sauter une ligne
        sauter trois espaces
Fin
```

2. Abstraction

```
// séquence avec fonction
écrire("paragraphe 1")
nouveauParagraphe()      // appel de la fonction
écrire("paragraphe 2")
```

Fonction

- Une fonction est une boîte noire, indépendante du reste du code, **effectuant une séquence d'instructions prédéterminée à chaque fois qu'elle est *appelée***.
- Une fonction peut ainsi être testée plus facilement.
- L'utilisateur d'une fonction n'a pas besoin de savoir ce qu'il y a dans la boîte noire, *ie.* comment elle a été codée.
- **Un *gros* programme se construit par décomposition en *petites* fonctions.**

Paramètres

Fonction modifiée

```
// définition de la fonction
Fonction nouveauParagraphe(nLines entier)
Début
        POUR i de 1 à nLines FAIRE
                sauter une ligne
        FIN_POUR
        sauter trois espaces
Fin
```

Souvent, le comportement d'une fonction doit être modulable selon des données fournies par l'utilisateur.

Dans l'exemple précédent, on peut vouloir que la fonction `nouveauParagraphe()` saute un nombre variable de lignes. Ce nombre sera donné par l'utilisateur lors de l'appel de la fonction.

```
Variables
        nbLignes entier = 6
Début
nouveauParagraphe(5)        //    on veut 5 sauts de ligne
nouveauParagraphe(nbLignes) // ... nbLignes sauts de ligne
Fin
```

Le nombre de lignes est alors un **paramètre** que l'on **passe** à la fonction.

Paramètres multiples

Le nombre de paramètres n'est pas limité.

```
// définition de la fonction
Fonction nouveauParagraphe(nLines entier, nSpaces entier)
Début
        POUR i de 1 à nLines FAIRE
                sauter une ligne
        FIN_POUR
        POUR i de 1 à nSpaces FAIRE
                sauter un espace
        FIN_POUR
Fin
```

Paramètres

- Les paramètres figurant dans la définition de la fonction sont les **paramètres formels**.
- La liste des paramètres formels équivaut à une déclaration de **variables locales** à la fonction.

Définition de la fonction

```
Fonction nouveauParagraphe(nLines entier, nSpaces entier) //nLines,nSpaces = param. formels
Début
        POUR i de 1 à nLines FAIRE
                sauter une ligne
        FIN_POUR
        POUR i de 1 à nSpaces FAIRE
                sauter un espace
        FIN_POUR
Fin
```

Appel de la fonction

```
nouveauParagraphe(nbLignes,3) // nbLignes et 3 = paramètres effectifs
```

Documentation

Une fonction doit être documentée : **Avant** sa définition, on trouve un bloc de commentaires comportant :

- Le nom et l'utilité de la fonction,
- éventuellement son fonctionnement en quelques mots (algo mis en oeuvre, ...),
- la liste des paramètres avec leur type et leur rôle dans le traitement,
- le type de retour et sa signification,

Fonction

Imbrication, appels et signature

- Une fonction peut en appeler d'autres.
- À chaque appel, les variables locales utilisées sont indépendantes de celles utilisées lors des appels précédents.
- Plusieurs fonctions peuvent porter le même nom. On distingue alors la fonction a appeler par sa **signature**.
- La **signature** d'une fonction comprend son nom ainsi que la liste des types de ses paramètres.

Exemples

```
Fonction nouveauParagraphe(nLines entier)
Fonction nouveauParagraphe(nLines entier, nSpaces entier)
```

Fonction: Terminaison et retour

- Les fonctions se terminent lorsque la dernière de leurs instructions a été exécutée.
- Une fonction peut aussi se terminer en **passant** une valeur à la séquence qui l'a appelée.
  - On dit qu'elle **retourne**, ou **renvoie**, une valeur.
  - La fonction se termine quelle que soit la position du retour dans la fonction.

Exemples

```
Fonction x_au_carre(réel x) retourne réel
Fonction multiplie(réel x, réel y) retourne réel
Fonction tire_au_sort() retourne booléen
```

```
Fonction estSuperieur(n entier, seuil entier) retourne booléen
Variables
        resultat booléen
        Début
                Si n > seuil ALORS resultat := vrai
                SINON resultat := faux
                FIN_SI
                RETOURNE resultat
        Fin
```

Fonction en Java

- Les fonctions, lorsqu'elles sont définies dans une classe, sont appelées **méthodes**.
- La méthode **main()** est une méthode *normale*, avec paramètres mais sans retour.
- La seule particularité de la méthode **main()** est d'être appelée automatiquement au lancement du programme.
- **Le type de retour** est indiqué avant le nom de la méthode

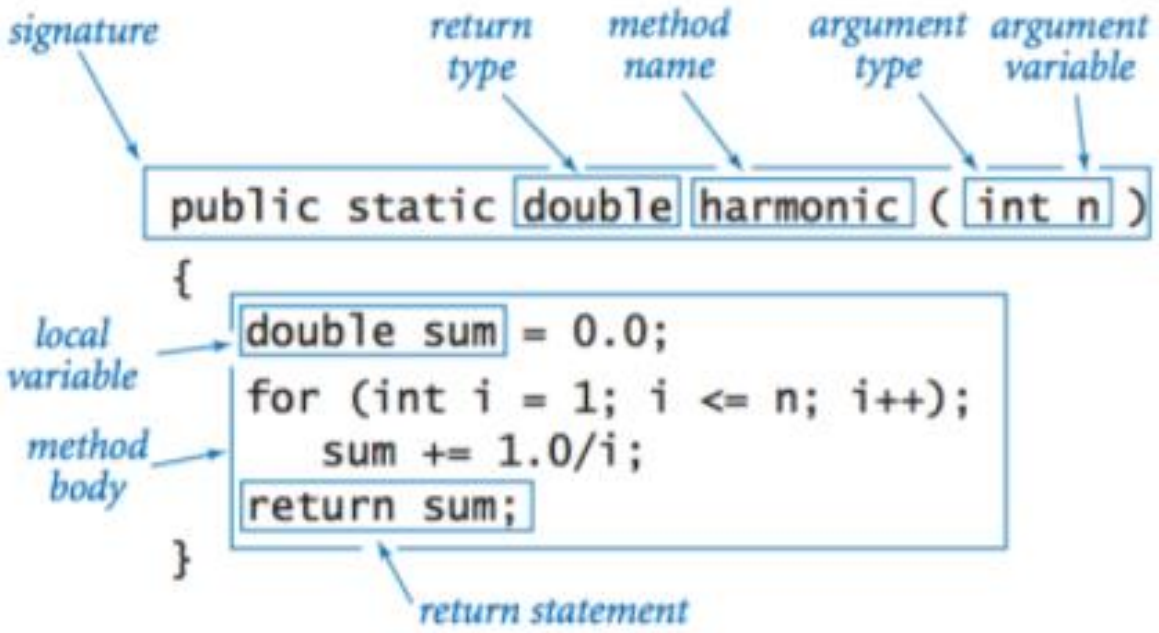
et **les paramètres entre parenthèses, séparés par des virgules.**

- Pour une méthode qui ne retourne rien, on précise **void** commme type de retour.

```
public static void nouveauParagraphe(nLines entier){
        for(int i=0; i< nLines; i++)
                System.out.println();
        System.out.println();
        System.out.println();
        System.out.println();
}
```

```
public static  boolean estSuperieur(int n, int seuil){
        if (n > seuil) return true;
        return false;
}
```

Fonction en Java



Noms des méthodes

La convention la plus fréquemment adoptée consiste à construire le nom d'une méthode en concaténant **les noms des mots qui décrivent son rôle**, **avec une majuscule à chaque première lettre de mot, sauf le premier**

Exemple

```
public void vaChercherLeBiscuit(){;}
public double calculeLaTaxe(double prixHT){;}
```

Exemples

```java
public class TestFonction0 {

  static void myMethod() {
    System.out.println("I just got executed!");
  }

  public static void main(String[] args) {
    myMethod();
    myMethod();
    myMethod();
  }
}
```

```java
public class TestFonction1 {

  static void myMethod(String fname, int age) {
    System.out.println(fname + " is " + age);
  }

  public static void main(String[] args) {
    myMethod("Liam", 5);
    myMethod("Jenny", 8);
    myMethod("Anja", 31);
  }
}
```