INF1256 Informatique pour les sciences de la gestion

– *Spécialisation des algorithmes et fonctions (méthodes)* –

Johnny TSHEKE, Ing. Jr.

Université du Québec à Montréal
Département d'Informatique
tsheke_shele.johnny@uqam.ca

Séance 08

Utilité des fonctions

- Réduire les duplications des codes (Pour réutiliser les codes)
- Faciliter la maintenance du code
- Faciliter la compréhension du code
- Découper le problème complexe en sous problèmes plus simple (sous programme)
- Se focaliser essentiellement sur une tâche spécifique
- ...

⇒ Intuitivement, une fonction est un nom donné à une séquence d'instructions. À chaque fois qu'on veut exécuter cette séquence d'instructions, on fait référence à ce nom.

⇒ En Java, une fonction est appelée Méthode. Dans la suite de ce cours nous parlerons donc de Méthode

Définition et appel des méthodes

Définition : C'est la partie du code qui crée la méthode. On l'appelle aussi déclaration de la méthode

Appel : c'est l'utilisation de la méthode (de son nom). On parle aussi de invocation de la méthode

$\Rightarrow$ Une méthode ne s'exécute que si elle est appelée

Principaux types des méthodes

- La méthode principale, généralement appelée main
- Un constructeur est comme une méthode mais porte le nom de la classe et ne retourne rien. Elle sert à créer des instances (objet) de la classe
- Les méthodes prédéfinies. ex : `chaine.length()`
- Les méthodes d'une librairie comme la classe Math
- Les méthodes définies par le programmeur
- ...

Forme générale pour définir une méthode

```
typeRetour nomMethode(paramètres){
    // Le corps de la methode
    return résultat
  }
```

- typeRetour type de la données à retourner. si `void` alors la methode ne retourne rien (pas de return)
- paramètres et return sont optionnels
- paramètres est une suite de déclaration de variables séparées par des virgules (ex : int nombre, String nom).
- nombre de paramètres variable → type dernier paramètre avec 3 points (ex : int... nombres) et traité comme tableau
- résultat est optionnel et ne peut-être présent que s'il y a return. résultat est en réalité l'argument de return
- nomMethode est le nom de la fonction.

⇒ Le corps de la méthode peut aussi contenir les appels d'autres méthodes.

Exemple d'une méthode simple : "Hello World"

Programme simple : HelloWorld.java

```
package s08;

public class HelloWorld {

        public static void main(String[] args) {
                /*
                 * Il faut un objet pour appeler une méthode non statique
                 * à partir d'une méthode statique
                 */
                HelloWorld h= new HelloWorld();
                h.hello();//appel de la méthode hello

        }

        void hello(){//méthode hello ne retourne rien car void
                System.out.println("Hello World!");
        }

}
```

Appel des méthodes

- Pour exécuter une méthode, il faut l'appeler
- On ne peut appeler qu'une méthode déjà définie (ou prédéfinie)
- Forme générale d'appel d'une fonction

```
nomMethode(arguments);
```

ou

```
varObj.nomMethode(arguments);
```

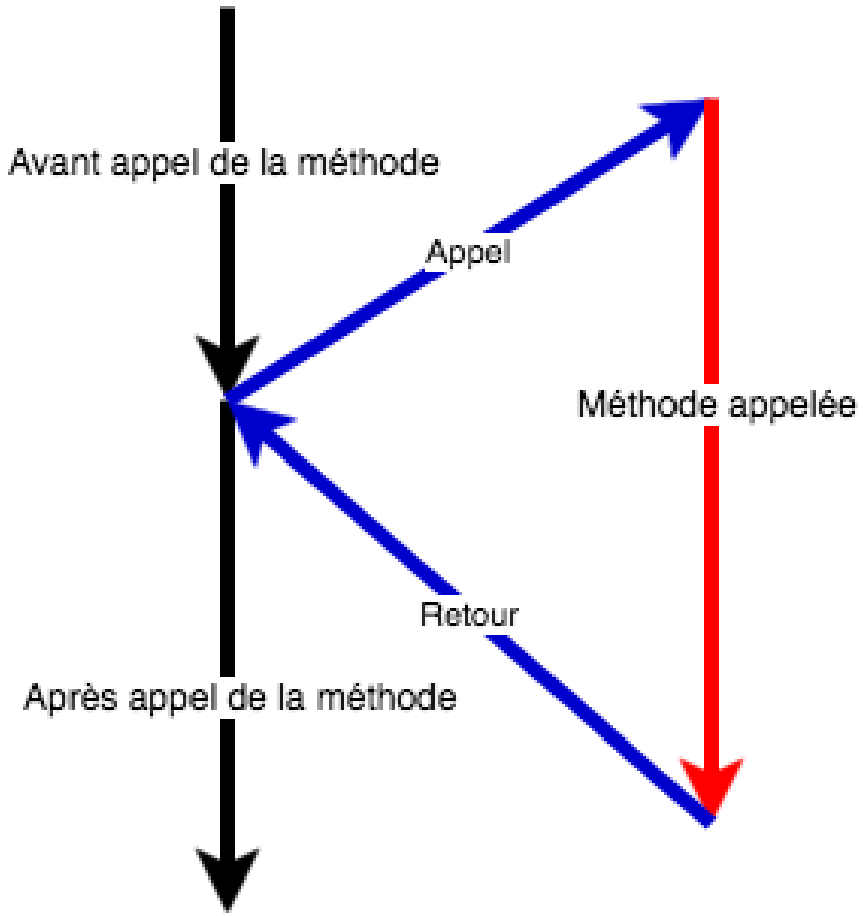
- On ne peut avoir arguments que si la méthode est définie avec paramètres
- Une méthode peut-être appelée à partir de la même classe (fichier .java) ou à partir d'une autre casse après importation de la classe dans laquelle elle est définie

Appel des méthodes (2)

En principe, à l'appel d'une méthode :

1. Le programme qui appelle se suspend
2. Les paramètres de la méthode appelée s'initialisent aux valeurs passées en arguments de l'appel
3. La fonction appelée s'exécute
4. On retourne au programme qui a appelé au point où il s'est suspendu
5. l'exécution du programme continu par la suite

⇒ Ici, on ne considère pas les exécutions parallèles



Exemple d'appel d'une méthode

Appel d'une méthode à partir d'une autre : helloWorld2.py

```
package s08;

public class HelloWorld2 {

        public static void main(String[] args) {
                /*
                 * Il faut un objet pour appeler une méthode non statique
                 * à partir d'une méthode statique
                 */
                HelloWorld2 h= new HelloWorld2();
                h.hello();//appel de la méthode hello

        }

        void hello(){//méthode hello ne retourne rien car void
                System.out.print("En Français: ");
                salut();//appel méthode salut
                System.out.println("In English: Hello World!");
        }
        void salut(){//méthode salut
                System.out.println("Salut le Monde!");
        }

}
```

Portée des variables

- La portée d'une variable déclarée dans une méthode (variable locale) est limitée dans la méthode
- Les paramètres d'une méthode ont une portée locale (comme les variables locales)
- Les variables et les constantes globales de la classe sont accessible dans les méthodes

Paramètres d'une méthode

- Un paramètre est une variable qui est initialisée à l'appel de la méthode
- Les paramètres sont des entrées de la méthode
- La portée des variables paramètres est limitée dans la méthode
- Pour avoir la valeur d'une variable déclarée dans une méthode dans une autre méthode, il faut passer la variable en argument.

$\Rightarrow$ les paramètres et les variables définies dans une méthode ne sont accessibles que dans la méthode

return dans une méthode

Lors de l'exécution d'une méthode,

- si on rencontre return, on arrête l'exécution de la méthode et on retourne le contrôle au point d'appel.
- Si on fourni une valeur en argument de return, alors cette valeur sera retourné au point d'appel. Type de la valeur retournée doit correspondre à celui indiqué dans la déclaration de la méthode

Exemple d'une méthode retournant une valeur

Appel d'une méthode (avec arguments) retournant une seule valeur : Calculs.java

```
package s08;
import java.util.*;
public class Calculs {

        public static void main(String[] args) {

                Calculs cal= new Calculs();
                double s = cal.somme(12.9,43.8,45.0);//remplacer . par , si nécessaire
        System.out.println("s ="+s);// peut afficher 101.699999 à cause de la précision ↙
      ↳ machine
        cal.affichage(s);
        }

        double somme(double... nombres){//nombre variable d'arguments type double
                double soe = 0.0;
                for(int i=0;i<nombres.length;i++){
                        soe = soe + nombres[i];
                }
                return (soe);
        }

        void affichage(double nombre){//methode avec 1 paramètre
                System.out.format("%nLa somme = %.3f %n",nombre);
        }

}
```

Saisie des données dans plusieurs

Si on doit saisir les données au clavier dans plusieurs méthodes, on peut :

- utiliser une variable globale de type Scanner Ce cas s'apprête mieux si tout se fait dans la même classe. La variable globale est accessible de partout. On peut l'instancier au début de la méthode `main` et fermer le scanner (.close()) à la fin de `main`
- passer un objet de type Scanner en paramètre des méthodes, Cette approche se prête mieux si on fait la saisie dans des classes différentes

Exemple de saisie de données dans plusieurs méthodes

Saisie de données dans plusieurs méthodes : SaisieDonnees.java

```
    Scanner clavier;
    public static void main(String[] args) {
            SaisieDonnees saisie = new SaisieDonnees();
            saisie.clavier = new Scanner(System.in);
            String pat = saisie.saisirPattern();
            System.out.println("Saisir une chaine respectant le pattern: "+pat);
            String entree = saisie.saisirChaine(pat);
            System.out.println("La chaine saisie est : "+entree);
    saisie.clavier.close();
    }

    String saisirPattern(){// ne fait pas de boucle
            System.out.println("Veuillez saisir un pattern -- Expression reguliere
↳ svp");
            String pat = "";
            try{
                    pat = clavier.next();
            }catch(Exception e){
                    pat=".";//valeur par défaut
            }
            return(pat);
    }
    String saisirChaine(String pattern){
            String chaine="";
            try{
                    chaine = clavier.next(pattern);
            }catch(Exception e){//
            }
            return(chaine);
    }
```