Yann.Secq@univ-lille.fr
Fabien.Delecroix@univ-lille.fr

TD n°5

Objectifs : Comprendre le flux d'exécution lors des appels de fonctions. Voir comment on peut réutiliser les fonctions qu'on a définies. Écrire ses premières fonctions utilisant une boucle à détection d'événement.

Définition et utilisation de fonctions

Rappel

Une fonction est un algorithme réutilisable dans différents contextes. Pour être réutilisé, cet algorithme indique précisément les informations nécessaires en entrée pour effectuer son traitement, ainsi que la nature (le type) du résultat qui est produit. La plupart des instructions que vous utilisez depuis le début du cours sont des appels de fonctions. Vous avez donc déjà une certaine maîtrise de l'utilisation de fonctions pré-existantes. Ici, on va se focaliser sur la création de ses propres fonctions et leur utilisation.

En `Java`, une signature de fonction respectera toujours la syntaxe suivante, qui donne le type du résultat, le nom de la fonction et la liste des paramètres :

`<type du résultat> nomDeLaFonction(<liste de paramètres>)`

La liste des paramètres est constituée d'autant de déclarations de paramètres (ou aucune !) que l'on souhaite, séparées par des virgules. Un paramètre peut être considéré comme une variable n'existant qu'au sein de la fonction. Il n'y a ainsi aucun sens à utiliser un paramètre en dehors de la fonction.

Voici un court exemple illustrant la définition puis l'utilisation d'une fonction qui calcule la circonférence d'un cercle (son périmètre).

```
1  class Circonference extends Program {
2      double circonference(double rayon) {
3          return 2*3.14159*rayon;
4      }
5
6      void algorithm(){
7          double rayon;
8          print("Entrez le rayon du cercle : ");
9          rayon = readDouble();
10         println("La circonférence du cercle est de : " + circonference(rayon));
11     }
12 }
```

Pour l'instant, vous ne pouvez mettre qu'**un seul `return`, toujours en dernière instruction de la fonction**.

Parfois, un algorithme ne retourne pas de résultat, souvent lorsqu'il produit des effets de bord (affichages par exemple). Dans ce cas, le type de retour de la fonction est `void`. On parle alors de procédure et il ne faut donc pas d'instruction `return` dans le corps de la procédure:

```
1  class TestProcedure extends Program {
2      void dessineLigne(int n) {
3          for (int i=0; i<n; i++) {
4              print("*");
5          }
6          println();
7      }
8      void algorithm() {
9          dessineLigne(10);
10     }
11 }
```

Attention : une fonction ne peut pas être déclarée à l'intérieur d'une autre fonction :)

Flux d'exécution et fonctions

Exercice 1 : Utilisation de fonctions

Considérez les deux programmes de la figure ci-dessous. Répondez aux questions qui suivent pour le premier programme :

1. Combien y a-t-il de définitions de fonctions dans le programme ?
2. Pour chaque fonction définie, indiquez sa signature et son corps.
3. Combien y a-t-il d'appels de fonctions ?
4. Pour chaque appel de fonction, indiquez le nom de la fonction appelée et les valeurs des arguments de l'appel.
5. Que se passe-t-il lors de l'exécution du programme ?

Mêmes questions pour le second programme. Que concluez-vous ?

```
1  class Exemple1 extends Program {
2    int plusCinq (int n) {
3      int x;
4      x = n + 5;
5      return x;
6    }
7    void algorithm () {
8      int n = 2;
9      int x = plusCinq(n);
10     println(x);
11   }
12 }
```

```
1  class Exemple2 extends Program {
2  1  int ajouter5 (int a) {
3  2    int x;
4  3    x = a + 5;
5  4    return x;
6    }
7  5  void algorithm () {
8  6    int x = 2;
9  7    int a = ajouter5(x);
10 8    println(a);
11   }
12 }
```

Exercice 2 : Les fonctions connues

Donner la liste de toutes les fonctions que vous avez déjà utilisées en TD ou en TP sans avoir à les définir: ce sont des fonctions prédéfinies dans la librairie `ap.jar`. Pour chaque fonction, donner sa signature.

Exercice 3 : Dessins de quadrilatères de caractères [FUN-DEF*, FUN-APPEL]

Dans cet exercice nous cherchons à générer des quadrilatères en caractères ASCII.

```
***      XXXX     OO
***      XXXX     OO
***               OO
                  OO
```

Dans cet exercice, pensez à réutiliser votre code.

— Qu'on dessine un carré ou un rectangle, on remarque qu'il s'agit de dessiner des suites de lignes de caractères de même taille. Proposez une procédure qui dessine une ligne d'une taille et d'un type de caractère passés en paramètres. Vous pouvez vous inspirez de l'exemple en début de TD. [FUN-DEF*]

— Proposez maintenant une procédure qui dessine des rectangles. [FUN-APPEL*, FUN-DEF]

— Proposez pour finir une procédure qui dessine des carrés. [FUN-DEF, FUN-APPEL]

— En incluant les fonctions réalisées, écrivez un programme qui permette les scénarios suivants : [FUN-APPEL]

```
Rectangle(1) ou Carré(2) : 1
Longueur : 5
Largeur : 3
Caractere : R
RRRRR
RRRRR
RRRRR
```

```
Rectangle(1) ou Carré(2) : 2
Taille : 4
Caractere : C
CCCC
CCCC
CCCC
```

Exercice 4 : Utilisation de ses propres fonctions [FUN-DEF, FUN-APPEL]

1. Écrire une fonction `char enMinuscule(char car)` qui correspond à la spécification suivante. Si `car` est une majuscule, alors `enMinuscule` retourne la minuscule correspondante. Sinon, `enMinuscule` retourne le caractère `car` lui même.
2. Écrire une fonction `String phraseEnMin(String phrase)` qui produit une chaîne de caractères contenant le même texte que `phrase`, mais toutes les majuscules sont transformées en minuscules. Pour produire le caractère à l'indice `indice` de la phrase résultat, vous utiliserez la fonction `enMinuscule` que vous avez déjà définie.
3. Écrivez enfin une fonction principale `void algorithm()` qui affiche la version en minuscule de chaque chaîne entrée saisie par l'utilisateur jusqu'à rencontrer *fin*.

Exercice 5 : Nombre de caractères par type [FUN-DEF, FUN-APPEL]

Voici un programme :

```
1  class NombreDeCaracteres extends Program{
2    void algorithm(){
3      String txt = readString();
4      int nbMaj=0, nbMin=0, nbChiffres=0;
5      for (int idx=0;idx<length(txt);idx=idx+1){
6        if (charAt(txt,idx)>='A' && charAt(txt,idx)<='Z'){
7          nbMaj=nbMaj+1;
8        }else if (charAt(txt,idx)>='a' && charAt(txt,idx)<='z'){
9          nbMin=nbMin+1;
10       }else if (charAt(txt,idx)>='0' && charAt(txt,idx)<='9'){
11         nbChiffres=nbChiffres+1;
12       }
13     }
14     println("Majuscules␣:␣" + nbMaj);
15     println("Minuscules␣:␣" + nbMin);
16     println("Chiffres␣:␣" + nbChiffres);
17   }
18 }
```

— Que fait-il ?

— Cet algortihme est plutôt dense et répète plusieurs fois des opérations similaires, proposez une version améliorée sur ces points via la création et l'utilisation d'une fonction d'une part et d'une variable d'autre part.