Université de Nice – Sophia Antipolis — Deug MASS MIAS SM 1re année
UEF1 : Informatique & Programmation — 2000–2001

TP N° 9

Modularité, localité et méthodes

Ça va mieux en le révisant ... L'exécution de chacune des classes possédant une méthode `main`(...) que l'on a écrites depuis le début de l'année ne s'est jamais bornée à l'exécution *stricto sensu* des instructions comprises dans sa méthode `main`(...). Nous avons d'emblée -et dans le désordre- fait appel à d'autres méthodes de la même classe, à des méthodes d'autres classes ainsi qu'à des méthodes de classes d'autres *packages*. C'est une telle organisation du code qui porte le nom de *modularité*.
Une classe appartient ou non à un *package* donné, une méthode est définie ou non dans une classe donnée, une variable est déclarée ou non dans une méthode donnée, une variable est déclarée ou non dans un bloc entre accolades donné. Tout ceci conduit aux notions d'*intérieur* (ou de *localité*) et d'*extérieur*.
Plaçons-nous au niveau des méthodes. De l'extérieur, il suffit de se documenter sur leur rôle et leur *signature* (leur nom et le type de leurs arguments, pris dans l'ordre) pour savoir comment les utiliser. Rappelez-vous les deux catégories de méthodes que sont les fonctions et les procédures. Une fonction doit se borner à renvoyer le résultat d'un calcul alors qu'une procédure doit posséder un seul effet de bord. Lors de l'appel à une méthode, donc de l'extérieur toujours, les paramètres sont *passés par valeurs*, c'est-à-dire que leurs valeurs sont recopiées dans des variables temporaires, à l'intérieur cette fois, portant le nom des *paramètres formels*[1]. Les variables qui sont des paramètres de départ, extérieurs, ne seront donc pas modifiées par l'appel d'une méthode. A l'intérieur d'une méthode, la portée d'une variable -locale donc- s'étend depuis sa déclaration jusqu'à la fin du bloc entre accolades dans lequel elle a été déclarée sauf dans le cas particulier de l'entête d'un `for`, où elle est locale à la boucle.

1 Effet de bord

Exercice 1) Voici une fonction douteuse calculant la factorielle d'un entier passé en paramètre :

```
static int factorielle (int n)
{  if ( n<0 )
    { System.out.print( "La factorielle d'un entier strictement "
      + "négatif n'est pas définie" ) ;
      return -1;
    }
   else if ( n==0 || n==1 )
      return 1;
   else
   {  int f=1 ;
      for ( int i=2 ; i<=n ; i=i+1 )
         f = f * i ;
      return f;
   }
}
```

1. Faites la trace des appels `factorielle(2)`, `factorielle(0)` puis `factorielle(`-1`)`.

1. Les paramètres formels sont ceux dont les noms figurent dans la définition d'une méthode : ce sont les paramètres locaux. Ils sont appelés ainsi car ils n'ont pas d'existence propre (ni de valeur!) en dehors des appels à la méthode.

2. En quoi cette méthode de classe `factorielle`(...) n'est pas acceptable ? Justifiez votre réponse.

◇

2 Fonctions ou procédures

Exercice 2) Il faut choisir : Inscrivez à bon escient des croix dans les cases du tableau ci-dessous :

Méthodes	Classe	Méth. de classe	Méth. d'instance	Fonctions	Procédures
`randomInt`(...)	`Numerik`				
`leNom`(...)	`Employé`				
`println`(...)	`Console`				
`translate`(...)	`Point`				
`length`(...)	`String`				
`pow`(...)	`Math`				

◇

Exercice 3) Nous allons étoffer la classe `Employé` créée au TD N° 4. Reprenez donc votre fichier du même nom que la classe.

1. Ajoutez à cette classe une méthode d'instance `affiche`(...) qui édite de façon tout à fait lisible (sur plusieurs lignes par exemple) les caractéristiques d'une instance `Employé`.
2. Créez une classe contenant une fonction `main`(...), de sorte à tester la méthode `affiche`(...) précédente. Faites-en sorte qu'elle utilise aussi la méthode `toString`(...).
3. En quoi la méthode `affiche`(...) ressemble-t-elle à la méthode `toString`(...) déjà présente dans cette classe ? En quoi en est-elle radicalement différente ?

◇

3 Paramètres de procédures

Exercice 4) Des méthodes erronées se sont glissées parmi les suivantes, mais lesquelles...

```
class TP9Ex4
{  public static void main (String [] args)
   { ...
   }
   static void echange(double x, double y)
   {  double tmp = x;
      x = y;
      y = tmp;
   }
   static void echangeEmployés(Employé x, Employé y)
   {  Employé tmp = x;
      x = y;
      y = tmp;
   }
   static void echangeSalaires(Employé x, Employé y)
```

```
   {  double tmp = x.leSalaire();
      x.fixeSalaire(y.leSalaire());
      y.fixeSalaire(tmp);
   }
}
```

1. Ajoutez à votre classe `Employé` une méthode `fixeSalaire`(...) qui permet de fixer le salaire d'une instance au nombre approché passé en paramètre.
2. Recopiez ce squelette de programme et remplissez les points de suspension de sorte à tester chacune des trois méthodes d'échange ci-dessus.
3. Les trois procédures d'échange proposées ont-elles l'effet escompté? Pourquoi? Assurez vous que vous avez bien compris le mécanisme de passage des paramètres par valeur... c'est très important.
4. Changez les procédures qui sont incorrectes en fonction retournant une valeur. Vous devrez de ce fait adapter la méthode `main`(...) aux nouveaux usages des nouvelles fonctions.

◇

Exercice 5) Aidez leurs chefs d'états à s'y retrouver quand ils se livrent à la comparaison des cinq pays mentionnés au tableau ci-dessous.

1. Définissez une classe `Pays` qui permette de stocker un nom de pays, accompagné de sa superficie et de sa population.
2. Ecrivez une autre classe destinée à lire les données des 5 pays. Elle doit créer autant d'instances de la classe `Pays` puis les afficher.
3. Ajoutez à la classe `Pays` précédente trois méthodes d'instance `estPlusVaste`(...), `estPlusPeuplé`(...) et `estPlusDense`(...), qui compare les caractéristiques concernées de deux pays.
4. Aménagez votre classe-test afin qu'elle affiche le pays à la population la plus nombreuse, celui à la plus grande superficie et enfin, celui à la plus grande densité de population à l'issue de la lecture des données.

Données:

Pays	Superficie (milliers de km^2)	Population (millions d'habitants)
Allemagne	256 800	79
Espagne	507 000	34
France	551 500	56
Italie	301 200	54
Royaume-Uni	244 000	56

◇

4 Approfondissement

Exercice 6) Pour cet exercice, nous vous présentons le cahier des charges, mais nous vous laissons libre de la réalisation.

1. Fabriquez une classe `NombresComplexes` qui offre toutes les opérations de calcul sur le corps des complexes.

2. Ajoutez un jeu de méthodes de plus à votre classe afin pouvoir visualiser ces nombres dans le plan complexe, lui-même mis dans une applet.

◇