université PARIS DIDEROT PARIS 7

INTRODUCTION À L'INFORMATIQUE ET À LA PROGRAMMATION (IF1)

TP2 : MÉTHODES ET FONCTIONS

Les méthodes et fonctions constituent un outil fondamental qui permet en particulier d'organiser un programme. À la fin de ce TP2, l'essentiel de ce qui aura été vu lors du TP1 deviendra obsolète : la fonction `main` ne devra désormais contenir que certaines instructions appelant les fonctions définies séparément, chaque fonction ayant un rôle bien déterminé.

1 Quatre fonctions moyennes et une fonction `main`

Certaines fonctions sont très proches dans leur principe aux fonctions mathématiques : elles prennent des valeurs numériques en arguments et renvoient une valeur numérique.

► **Exercice 1** : Soit un rectangle de longueurs de côté a et b. La longueur de côté du carré ayant

- même périmètre est la moyenne arithmétique $\frac{a+b}{2}$
- même aire est la moyenne géométrique $\sqrt{ab}$
- même rapport aire sur périmètre est la moyenne harmonique $\frac{2}{\frac{1}{a}+\frac{1}{b}}$
- même diagonale est la moyenne quadratique $\sqrt{\frac{a^2+b^2}{2}}$

1. La fonction `sqrt` calcule la racine carrée. Rechercher le nom de la classe où elle est définie. Donner son en-tête.
2. Écrire quatre fonctions renvoyant les différentes moyennes rappelées ci-dessus.
3. Écrire une fonction `main` qui demande les longueurs (réelles) des côtés d'un rectangle et affiche quatre messages, chacun de la forme : `La longueur du cote du carre ayant meme ... est ...`

2 Chercher l'information pour construire une fonction

L'analyse méthodique d'un énoncé de problème permet de dégager les éléments nécessaires à la construction des fonctions qui permettront de répondre au problème en question.

► **Exercice 2** : Dans un bureau, il y a une fenêtre aluminium à rideaux rouges. La fenêtre a une hauteur de 1.60m et une largeur de 1.20m. Les rideaux font 60cm de largeur chacun, un à gauche, l'autre à droite. Ils sont tirés horizontalement et nécessairement de façon symétrique et sans plis. On souhaite construire une fonction `ensoleillement` qui renvoie le pourcentage (entier) de lumière qui passe en fonction de la position des rideaux. Par exemple, si les rideaux sont fermés de 0, alors on a 100% de lumière, si les rideaux sont tirés de 60, alors on a 0% de lumière. On supposera que le soleil diffuse dans l'axe de la fenêtre.

1. Récapituler sur un schéma les données nécessaires au calcul de l'ensoleillement.
2. Distinguer les données fixes (constantes) et les données variables (paramètres ou arguments). Leur donner des noms adaptés. Leur attribuer des types adaptés.
3. Donner un nom et un type adaptés à l'élément qui devra être calculé et renvoyé par la fonction.
4. Écrire l'en-tête de la fonction, puis son corps.
5. Écrire un programme permettant de la tester.

► **Exercice 3** : Reprendre chacun des énoncés et des codes des exercices **5**, **6**, **7**, **8**, **9** et **10** de la feuille TP1 avec un oeil neuf : écrire des fonctions et les utiliser dans la fonction `main`.

► **Exercice 4** : Reprendre l'énoncé de l'exercice **13** de la feuille TP1 avec un oeil neuf. Écrire une fonction `precision` qui prend en arguments un nombre réel `x` et un nombre entier `k` et renvoie la valeur arrondie de `x` à `k` chiffre(s) après la virgule. L'utiliser dans les codes des exercices **5**, **6**, **7** et **8**.

► **Exercice 5** : Reprendre chacun des énoncés des exercices **12**, **15** et **16** de la feuille TP1 avec un oeil neuf. Écrire les fonctions `jetDeDe`, `sontOrdonnes` et `estBissextile` correspondantes et les tester dans la fonction `main` de chaque classe.

► **Exercice 6** : On s'intéresse aux résultats d'une élection dans un scrutin de listes dans lequel il y a `ns` sièges à pourvoir et trois listes candidates ayant obtenu respectivement `v1`, `v2` et `v3` voix.

```
> java Scrutin
Combien de sieges? 50
Combien de voix pour la liste 1? 765765
Combien de voix pour la liste 2? 879873
Combien de voix pour la liste 3? 636573
Liste ayant obtenu 765765 voix: 16 sieges
Liste ayant obtenu 879873 voix: 19 sieges
Liste ayant obtenu 636573 voix: 13 sieges
Restant a pourvoir : 2 sieges
```

1. Écrire une fonction `nombreDeSieges` qui demande et récupère le nombre de sièges.
2. Écrire une fonction `nombreDeVoix` qui prend en argument un numéro `x` et demande et récupère le nombre de voix obtenu par la liste `x`.
3. La similarité du corps des deux fonctions précédentes invite à écrire une fonction `entierLu` qui prend en argument un message `m` et demande via le message `m` un entier et le récupère.

   Écrire une nouvelle version des fonctions `nombreDeSieges` et `nombreDeVoix` utilisant `entierLu`.
4. Écrire une fonction `quotientElectoral` qui prend en arguments le nombre `ns` de sièges et le nombre de voix de chacune des listes et renvoie le quotient de la division du nombre de voix total par `ns`.
5. Écrire une fonction `nombreDeSiegesListe` qui prend en arguments le quotient électoral `qe` et le nombre `vx` de voix d'une liste et renvoie le quotient de la division de `vx` par `qe`.
6. Écrire une fonction `scrutin` (sans argument) qui renvoie une chaîne de caractères décrivant le résultat d'un scrutin.

3 Surcharger

Un même identificateur peut être utilisé pour plusieurs fonctions pourvu que les signatures soient différentes.

► **Exercice 7** : Dans une même classe, écrire deux fonctions `triple` qui l'une prend un argument entier et renvoie son triple, tandis que l'autre prend un argument chaîne de caractères et renvoie sa triple concaténation. Ainsi `triple(79)` renvoie `237` et `triple("79")` renvoie `"797979"`. Tester le tout.

► **Exercice 8** : Dans la classe créée à l'exercice **1**, écrire quatre nouvelles méthodes ayant trois arguments réels.

1. Opter pour une approche directe : généraliser à trois dimensions les formules données pour deux dimensions. Tester les huit méthodes coexistantes.
2. Opter pour une approche Shadok[1] : pour la moyenne arithmétique, considérer l'égalité $\frac{a+b+c}{3} = \frac{\frac{4}{3}\frac{a+b}{2}+\frac{2}{3}c}{2}$ pour utiliser deux appels à la fonction à deux arguments dans la définition de la fonction à trois arguments. Vérifier dans quelle mesure cette approche Shadok est envisageable pour les trois autres moyennes.

   Tester les huit méthodes coexistantes.

1. Pourquoi faire simple, quand on peut faire compliqué !

4 Faire du golf

Code golf est un genre de compétition de programmation récréative où les participant.e.s s'efforcent d'atteindre le code le plus court possible pour implémenter un algorithme donné.

► **Exercice 9** : On veut construire puis afficher la chaîne de caractères correspondant aux paroles de la comptine *Savez-vous planter les choux ?*

1. Écrire les fonctions `refrain` et `couplet` correspondantes.
2. Écrire alors la fonction `comptine`.
3. Proposer d'autres possibilités de fonctions permettant de raccourcir le code.

5 Documenter un code

La *documentation* d'un code est destinée aux futurs utilisateurs de ce code. Il ne s'agit pas de commentaires liés aux détails de programmation [2]. Cette documentation doit permettre d'utiliser le code sans avoir à le regarder.
En `Java`, cette documentation peut être mise en place *dans* le code au moyen de commentaires entre `/**` et `*/` et d'une syntaxe appropriée. Ces commentaires permettent de générer une documentation au format `html` en utilisant la commande `javadoc` avec le nom du fichier source en argument. À noter que par défaut, `javadoc` ne génère que la documentation des classes publiques : en attendant de savoir ce que cela signifie, on ajoutera simplement le mot-clé `public` devant les en-têtes des classes.
Tous les documents à récupérer sur `didel` se trouvent dans le répertoire `tp2` de `Documents` et `liens`.

► **Exercice 10** :

1. Recopier le fichier suivant :
```
/**
 * classe inutile
 */
public class Inutile{
    /**
     * fonction qui ne fait rien
     * @param n un entier quelconque
     * @return n
     */
    public static int neFaitRien(int n){
        return n;
    }
}
```
2. Lancer la commande `javadoc` sur ce fichier : quels sont les fichiers d'extension `html` créés ?
3. Tous ces fichiers ont été créés dans le répertoire courant, ce qui rend peu lisible son contenu. Nettoyer son répertoire courant et y créer un sous-répertoire `docInutile`. Relancer la commande `javadoc` pour que les fichiers soient créés dans ce sous-répertoire grâce à l'option `d` :
   `javadoc -d docInutile Inutile.java`
4. Regarder le fichier `docInutile/index.html` dans un navigateur web et naviguer. Bien regarder où se trouve le texte qui était mis en commentaire dans le code source. À quoi sert le `@return` dans ces commentaires ?
5. On remarque que dans la documentation, la classe `Inutile` est présentée comme "provenant" de la classe `Object` (on dit qu'elle *étend* la classe `Object` ; ce qui est le cas de toutes les classes en `Java`), mais il n'y a pas de lien vers la classe `Object` dans la documentation. Pour y remédier, recréer la documentation en utilisant l'option `link` avec comme argument l'url de la doc officielle :
   `javadoc -d docInutile -link http://docs.oracle.com/javase/6/docs/api/ Inutile.java`
   Regarder à nouveau le fichier `docInutile/index.html` dans un navigateur web et constater que des liens ont été ajoutés.

2. Ces commentaires sont très importants pour permettre modifications et maintenance du code : ne pas les négliger !

On doit prendre l'habitude de documenter ses classes et ses fonctions au format standard `javadoc`, cela permet d'avoir aisément une documentation agréable et facilite la réutilisation d'anciens codes dans lesquels on n'a pas forcément envie de se replonger.

► **Exercice 11** : La classe `java.lang.String` permet de représenter des chaînes de caractères. Regarder sa documentation pour répondre aux questions ci-dessous et faire des tests sur machine :

1. Comment convertit-on une chaîne de caractères en la même avec des lettres majuscules ?
   exemple : `"ChaIne_23"` → `"CHAINE_23"`
2. Comment teste-t-on si une chaîne est le début d'une autre ?
   exemple : `"tot"` est le début de `"toto"`, mais pas de `"titi"`
3. Comment obtient-on une chaîne de caractères à partir d'un entier ?
   exemple : `123` → `"123"`

► **Exercice 12** : Télécharger sur `didel` les documents suivants du sous-répertoire `exercice12` : `img.jpg` et `librairieTP2.jar`, et les placer dans son répertoire de travail. Le fichier `librairieTP2.jar` est une *archive java* contenant la définition de plusieurs classes, dont `FenetreTP2` et `ImageTP2` qui seront utilisées dans la suite de l'exercice.
Suivre maintenant le lien `javadoc.url` (toujours au même endroit sur didel) : c'est la documentation des classes `FenetreTP2` et `ImageTP2`.

1. Regarder la documentation de la classe `ImageTP2` : quelle méthode permet d'obtenir une instance de `java.awt.image.BufferedImage` à partir du nom d'un fichier représentant une image (c'est-à-dire quelle méthode prend en argument le nom d'un fichier supposé contenir une image et renvoie une `BufferedImage`) ?
2. Regarder la documentation de la classe `FenetreTP2` : quelle méthode permet d'ouvrir une fenêtre graphique contenant l'image fournie en argument sous la forme d'une `BufferedImage` ?
3. Écrire un programme `Java` qui permet d'afficher l'image `img.jpg` à l'écran en combinant les deux méthodes précédentes. Pour signifier au compilateur java qu'on veut utiliser la librairie `librairieTP2.jar` pour compiler `MaClasse.java`, on utilise la ligne de commande suivante :
   `javac -classpath librairieTP2.jar MaClasse.java`

► **Exercice 13** (inspiré du partiel 2004/2005, exercice 4) : Télécharger sur `didel` le fichier suivant du sous-répertoire `exercice13` : `GrilleTP2.jar`. La documentation des classes de cette librairie est disponible au même endroit en suivant le lien `javadoc.url`.
Pour toutes les questions qui suivent, il s'agit d'écrire le code *et de documenter* les fonctions. Dans une même classe :

1. Écrire une fonction qui teste si un point est sur une grille (c'est-à-dire renvoie `true` s'il est dessus et `false` sinon).
2. Écrire une fonction qui teste si un point est sur la bordure d'une grille.
3. Écrire une fonction qui teste si un point est un des coins de la grille.
4. Écrire une fonction qui teste si trois points sont alignés.
   Rappel : 3 points $A(x_A, y_A)$, $B(x_B, y_B)$ et $C(x_C, y_C)$ sont alignés si et seulement si les vecteurs $\overrightarrow{AB}(x_{AB} = x_B - x_A, y_{AB} = y_B - y_A)$ et $\overrightarrow{AC}(x_{AC}, y_{AC})$ sont colinéaires, c'est-à-dire si et seulement si $x_{AB}y_{AC} = x_{AC}y_{AB}$.
5. Générer la documentation de cette classe.