TP 2 : Classes et tableaux

1 L'arbre généalogique

Dans cette partie, nous allons reprendre la classe Personne vue au TD2 et au TD3.

Exercice 1: Ecrire le code de la classe Personne (du TD2 au TD3).

Exercice 2: Ecrire le code du programme proposé à l'exercice 3 du TD3: impression de l'arbre généalogique.

2 Master Mind

Notre objectif est de réaliser un petit programme Master Mind. L'ensemble des fonctions suivantes doivent être placées dans un même fichier, de nom MasterMind.java (et donc définies dans une classe de nom **MasterMind**).

Exercice 1: Ecrire une classe MasterMind qui crée une instance de MasterMind au démarrage du programme:

```
class MasterMind {

                public static void main(String[] args){
                                MasterMind m = new MasterMind();
                }

}
```

Exercice 3: Dans la classe MasterMind, créer une méthode

boolean verifierTab(char[] tab)

qui analyse le tableau **tab** et retourne true si le tableau est de dimension 5 et qu'il ne contient que des lettres comprises entre 'A' et 'J', et retourne false sinon.

Exercice 4: Ecrire une méthode **principale()** qui lit, en boucle infinie, une chaîne saisie par l'utilisateur. On sort de la boucle seulement si la chaîne saisie est la chaîne "quit".

Chaque chaîne entrée est transformée en un tableau de caractères (rappel: *String*.toCharArray()), la méthode **verifierTab** doit ensuite être appliquée à ce tableau et le programme doit afficher : "chaine valide" si **verifierTab** a renvoyé true, "chaine non valide" sinon.

Exemple :
```
> java MasterMind
AEGCD
chaine valide
AEBBDC
chaine non valide
quit
>
```

Le programme principal doit maintenant appeler la méthode **principale** : m.principale();

Exercice 5: Ecrire une méthode **bienPlace** qui prend comme argument deux tableaux de caractères et qui retourne le nombre de caractères identiques placés à la même position dans les deux tableaux.

Exercice 6: Ecrire une méthode **malPlace** qui prend comme argument deux tableaux de caractères et qui retourne le nombre de caractères présents à la fois dans les deux tableaux, mais à des positions différentes.

Exemple: si t1={ A, F, F, G, D, A} et t2={ F, A, F, I, D, B}

malPlace(t1,t2) renvoie 2

Exercice 7: Modifier la méthode **principale** de l'exercice 4 et y ajouter une variable **`char[] solution`** initialisée avec 5 lettres comprises entre 'A' et 'J'. A la place de "chaine valide" le programme doit maintenant afficher :

" [*x*] bien-placé(s), et [y] mal-placé(s) "

où [*x*], respectivement [*y*], est le nombre d'éléments bien-placés, respectivement mal-placés, dans la chaîne saisie par l'utilisateur par rapport à la solution. Si l'utilisateur trouve les 5 bons caractères, le programme s'arrête en affichant: "vous avez gagné ! (Solution: [*la solution*])"
Et si l'utilisateur saisit "quit", le programme s'arrête en affichant :

"vous avez perdu ! (Solution: [*la solution*])"

Exercice 8: Modifier le programme de sorte que la solution initiale soit générée aléatoirement.

NB: la fonction: **public double random()** , de la classe Math, retourne une valeur décimale aléatoire comprise dans l'intervalle [0,0 ; 1,0[.