Corrigé du TD de Java n°2

1 PETITE MISE EN ROUTE

Dans cette partie, il faut réfléchir aux entêtes de certaines fonctions (quelles seront les paramètres de la fonction) et à leur valeur de retour. C'est un exercice crucial qui vous permettra, ensuite, de facilement poser les bases d'un programme avant de l'écrire sur votre feuille (ou sur l'ordinateur) :

1. Une fonction qui retourne le plus grand élément d'un tableau de double.

La fonction retourne un élément du tableau, donc un double, et prend en paramètre le tableau.

```
public static double plus_grand_element(double[] t)
```

2. Une fonction qui trie un tableau d'entiers.

La fonction prend simplement en paramètre le tableau qu'elle triera

```
public static void trier(int[] t)
```

3. Une fonction qui calcule la racine carrée d'un entier.

La fonction renvoie une racine carrée, donc un double, et prend en paramètre un entier.

```
public static double racine(int x)
```

4. Une fonction qui fusionne deux tableaux entre eux.

La fonction prend en paramètre deux tableaux (on ne sait pas si ce sont des int, double, ou autre... on va dire que ce sont des int) et renvoie un nouveau tableau.

```
public static int[] fusion(int[] t1, int[] t2)
```

5. Une fonction qui renvoie la moyenne des éléments d'un tableau d'entiers.

La fonction renvoie une moyenne, donc un double, et prend en paramètre un tableau.

```
public static double moyenne(int[] tabulo)
```

6. Une fonction qui calcule un entier élevé à une puissance entière.

La fonction prend en paramètre un entier x et un entier n qui sera la puissance à laquelle élever x, et renvoie un entier.

```
public static int puissance(int x, int n)
```

7. Une fonction qui recherche un élément particulier dans un tableau d'entiers.

La fonction prend en paramètre un tableau, un entier à rechercher, et renvoie un booléen permettant de savoir si oui ou non on a trouvé l'élément dans le tableau.

```
public static boolean rechercher(int[] tab, int x)
```

8. Une fonction qui affiche la décomposition en facteurs premiers d'un nombre.

La fonction prend en paramètre un entier, et ne renvoie rien car tout sera affiché à l'écran.

```
public static void decomposition(int x)
```

9. Une fonction qui renvoie la décomposition en facteurs premiers d'un nombre.

Pareil que précédemment, sauf qu'il faut renvoyer la décomposition... on choisira donc un tableau d'entiers pour renvoyer la décomposition en nombres premiers.

```
public static int[] decomposition(int x)
```

10. Une fonction qui fusionne deux cartes Pokemons, pour en faire une nouvelle super carte Pokemon !

La fonction prend deux CartePokemon en paramètre, et en renvoie une nouvelle.

```
public static CartePokemon fusion(CartePokemon p1, CartePokemon p2)
```

2 ÉLABORER UN STRUCTURE DE DONNÉES

Dans cet exercice, vous devez simplement élaborer une structure de données répondant aux questions.

1.On souhaite élaborer une structure de données permettant de suivre un élève. On souhaite y ranger le nom de l'élève, son prénom, ses notes en informatique, en maths et en anglais.

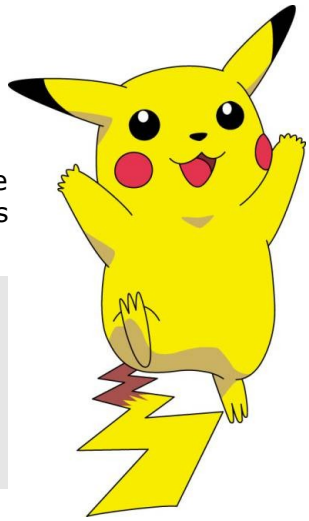
```
class Eleve
{
    String nom;
    String prenom;
    double note_anglais;
    double note_info;
    double note_maths;
}
```

2. On souhaite maintenant gérer une classe d'élèves. On veut stocker le nombre d'élèves, la promotion (l'année de sortie si tout va bien), et des élèves.

On va utiliser un tableau d'Eleve pour gérer "les élèves" dont il est question.

```
class Classe
{
    int promo;
    int nb_eleve;
    Eleve[] tous_les_eleves;
}
```

3 SAUVEZ-LES TOUS !



On continue le TD n°1 sur les Pokemons. On décide que la structure de données Pokemon contiendra un champs Pokemon, permettant de lier les Pokemons les uns aux autres...

```
class Pokemon
{
      String nom;
      String type;
      int puissance;
      Pokemon suivant;
}
```

De cette manière, on créé ce qu'on appelle une liste chaînée, c'est à dire un ensemble d'éléments liés les uns aux autres. Chaque élément de la liste (ou de la chaîne) est appelé un maillon.

1. Quel élément permet de retrouver tous les autres éléments de la liste ?

Le premier élément permet de retrouver tous les autres (en passant de suivant en suivant). On appelle cet élément la tête : il se caractérise par le fait qu'aucun autre élément de la liste ne "pointe" sur lui (aucun suivant ne permet d'y accéder).

Imaginez que vous possédez une liste de Pokemons.

2. Comment supprimer un élément de cette liste (réfléchissez à la donnée d'entrée) ?

La donnée d'entrée pourrait être un Pokemon ou bien une position de l'élément à supprimer. On parcourt la liste du début à la fin, et dès que l'on arrive avant le Pokemon voulu, on le supprime en le retirant de la liste.

```
public static void supprimer(Pokemon tete_liste, int position)
{
    Pokemon parcours;

    //Attention: ce code ne fonctionne pas si la liste ne possede plus q'un
    //élément... Dans ce cas, c'est plus compliqué.

    if(position==0) //il faut supprimer le premier element...
    {               //on va copier son suivant sur lui-meme, et supprimer
                    //plutot le suivant.
        //on copie le suivant sur la tete
        tete_liste.nom = tete_liste.suivant.nom;
        tete_liste.puissance = tete_liste.suivant.puissance;
        tete_liste.type = tete_liste.suivant.type;

        //on supprime le suivant
        tete_liste.suivant = tete_liste.suivant.suivant;
    }
    else
    {
        //On va se positionner juste avant l'élément à supprimer
        parcours = tete_liste;
        for(int i=0; i!=position-1; i=i+1)
        {
```

```
            parcours = parcours.suivant;
        }

        //et on supprime l'élément suivant de parcours
        parcours.suivant = parcours.suivant.suivant;
    }
}
```

3. Et si la liste est triée ?

Ca ne change rien.

4. Comment ajouter un élément à la liste ?

Il nous faut la (tête de la) liste, et un Pokemon à rajouter... On va dire que l'on rajoute l'élément à la fin de la liste.

```
public static void ajouter_a_la_fin(Pokemon tete_liste, Pokemon a_rajouter)
{
    Pokemon parcours = tete_liste;
    while(parcours.suivant != null)
    {
        parcours=parcours.suivant; //on avance jusqu'à la fin de la liste
    }

    //puis on rajoute notre Pokemon à la fin de la liste
    parcours.suivant = a_rajouter;
    a_rajouter.suivant=null;
}
```

5. Comment connaître la taille de la liste ? Voyez-vous un moyen plus simple pour obtenir la taille de la liste (qui inclurait de rajouter une structure de données) ?

Pas le choix pour le moment, il faut parcourir tous les éléments de la liste du début à la fin, et compter combien il y a d'éléments...

```
public static int taille_liste(Pokemon tete_liste)
{
    int taille=0;
    Pokemon parcours = tete_liste;

    while(parcours != null)
    {
        parcours=parcours.suivant; //on avance jusqu'à la fin de la liste
        taille = taille+1;
    }

    return taille;
}
```

On pourrait rajouter une structure de donnée GestionnaireListe qui contiendrait le premier élément de la liste et la taille de la liste... Toutes les fonctions d'ajout/suppression de Pokemon ne prendraient plus la tete de la liste en paramètre, mais cette nouvelle structure de donnée. Dès que l'on rajouterait ou supprimerait un Pokemon de la liste, on modifierait la taille de la liste dans cette structure. Si l'on souhaite connaître la taille de la liste, on se réfère directement à la variable.

```
class GestionnaireListe
{
```

```
    int taille_liste;
    Pokemon tete_liste;
}
```

6. Comment accéder (et renvoyer en sortie) au i° élément de la liste (par exemple comment accéder aux 4° élément de la liste) ? Et si la collection de Pokemons n'était pas une liste mais un tableau ?

Pas le choix, il faut parcourir la liste du premier au sixième élément. Dans un tableau tab, on ferait tab[i], ce qui est beaucoup plus rapide.

```
public static Pokemon acces(Pokemon tete_liste, int position)
{
    Pokemon parcours=tete_liste;

    for(int i=0; i!=position; i=i+1)
    {
        parcours=parcours.suivant;
    }
    return parcours;
}
```