TP n°6 Java

L'objectif de ce TP est de maîtriser les bases de l'héritage via l'écriture de quelques classes simples permettant de représenter des légumes, avec lesquels on peut faire de la soupe.
Les conventions de nommage des classes, des attributs, des méthodes et de leurs paramètres, des instances sont à prendre en compte.
Tous les attributs doivent respecter le principe d'encapsulation.
Des constantes de classe sont à utiliser (pas de nombre écrit en chiffres dans le code).
Utiliser deux packages différents pour les deux parties.

Exercice 1 :

Partie 1:

1. Ecrire une **classe Carrot** représentant une carotte avec son poids en kilogrammes et sa longueur en centimètres.
2. Ecrire une **Potatoe** représentant une pomme de terre avec son poids en kilogrammes et un nombre de germes ("yeux").
3. La pomme de terre ci-dessus a beaucoup de germes (10 pour un poids de 300 grammes). En règle générale, les pommes de terre ont un germe par tranche de 100 grammes. Ajoutez à la **classe Potatoe** ce qu'il faut pour créer des pommes de terre qui respectent cette règle générale sans avoir à spécifier explicitement le nombre de germes.
4. Ecrire le code nécessaire afin que les instructions suivantes produisent les affichages indiqués.

```
Carrot c = new Carrot(0.25, 30)
Potatoe pdt1 = new Potatoe(0.3, 10);
Potatoe pdt2 = new Potatoe(0.3);
System.out.println(c);
System.out.println(pdt1);
System.out.println(pdt2);
```

Résultat d'exécution :

```
Carotte: [0.25 kg - 30 cm]
Patate: [0.3 kg - 10 yeux]
Patate: [0.3 kg – 3 yeux]
```

Partie 2:

1. Comme on risque d'avoir plein de légumes (carottes, pommes de terre, poireaux...) mais que tous auront un poids, on souhaite créer une **classe Vegetable** pour "f**actoriser**" cet attribut. Modifier le code précédent afin que les instructions suivantes produisent les affichages indiqués.

Code main :	**Résultat d'exécution :**
Vegetable pdt = new Potatoe(0.3, 10); Vegetable c = new Carrot(0.25, 30); System.out.println(pdt); System.out.println(c);	Patate: [0.3 kg - 10 yeux] Carotte: [0.25 kg - 30 cm]

2. On veut maintenant définir une classe Soup représentant une soupe de légumes par l'ensemble de ses ingrédients (les légumes de type **Vegetable**). Utiliser une liste (ArrayList) afin de stocker les légumes. Une fois qu'une soupe est créée, on lui ajoute chaque légume un par un, grâce à la méthode **add** oomme illustré ci-dessous. A tout moment, on peut demander l'affichage de la soupe.

3. Toujours dans la **classe Soup**, on souhaite avoir une méthode **getPeelingWeight()** qui donne le poids des épluchures générées lors de la préparation de cette soupe.
   Par défaut, le poids d'épluchure des légumes est de 10% de leur poids. Néanmoins, dans le cas particulier des pommes de terre, le poids d'épluchure est ce pourcentage (10%) auquel il faut ajouter 10 grammes par germe (oeil).
   Écrire la classe **Soup** de telle sorte que les instructions suivantes produisent les affichages indiqués.

Code main :	**Résultat d'exécution :**

Soup s = new Soup();
s.add(new Potatoe(0.3, 10));
s.add(new Carrot(0.25, 30));
s.add(new Potatoe(0.500));
System.out.println(s);
	Soupe
	- - - - - -
	Patate: [0.3 kg - 10 yeux]
	Carotte: [0.25 kg - 30 cm]
	Patate: [0.5 kg - 5 yeux]

4. Ecrire le code nécessaire afin que les instructions suivantes produisent les affichages indiqués.

   ```
   Soup s = new Soup();
   s.add(new Potatoe(0.3, 10));
   s.add(new Carrot(0.25, 30));
   System.out.println(s.getPeelingWeight()); // affiche: 0.155
   ```

   0.155 : correspond à 25g pour la carotte (10% de 0,250kg) et à 130g pour la pomme de terre (30g = 10% de 0,300kg + 100g pour 10g fois 10 germes).
   Indication : utiliser la redéfinition de méthode

5. Ajouter à la classe Soup une méthode **sortByWeight()** qui permet de trier la liste de légumes selon leur poids.
   Indication : utiliser la méthode **Collections.sort(List)**.

Exercice 2 : Interface Condition

L'objectif de ce programme est de pouvoir compter combien d'éléments d'une collection vérifient ou ne vérifient pas un certain nombre de conditions.
1) Créer une interface Condition avec une seule méthode booléenne verif qui prend un entier en paramètre.
2) Créer une classe Pair implémentant l'interface Condition sans attribut.
   - Ecrire le constructeur par défaut.
   - Redéfinir la méthode toString afin qu'elle retourne entre parenthèses "Pair".
   - Définir la méthode devant vérifier que le nombre passé en paramètre est pair.

3) Créer une classe SuperieurA implémentant l'interface Condition et possédant un attribut entier ref qui est la valeur de référence avec laquelle il faut comparer chaque nombre à vérifier.

- Ecrire le constructeur à un paramètre adéquat.
- Ecrire l'accesseur en lecture getRef.
- Redéfinir la méthode toString afin qu'elle retourne entre parenthèses "SuperieurA" suivi de la valeur de référence.
- Définir la méthode devant vérifier que le nombre passé en paramètre est strictement supérieur à la valeur de référence.

5) Créer une classe TestConditions sans attribut qui ne contiendra que des méthodes de classe statiques.

- Ecrire la méthode remplitAlea devant remplir la liste (ArrayList), passée en 1er paramètre, avec n valeurs entières aléatoires dans l'intervalle [1, 50]. L'entier n est le second paramètre de cette méthode ne retournant aucune valeur.

- Ecrire la méthode compte retournant le nombre d'éléments de la liste, passée en 1er paramètre, qui vérifient la condition passée en second paramètre, si le booléen passé en troisième paramètre vaut true. Si ce booléen vaut false, la méthode retourne le nombre d'éléments de la liste ne vérifiant pas la condition.

• Ecrire la méthode main respectant les points suivants:

d) créer la liste d'entiers et la remplir aléatoirement avec n nombres
e) créer un tableau de conditions initialisé avec les 2 conditions telles qu'un entier v doit vérifier:

- v est pair
- v > 25

f) parcourir ce tableau de conditions et afficher pour chaque condition sa représentation sous la forme d'une String suivi de : , puis du nombre d'éléments vérifiant (respectivement ne vérifiant pas) cette condition, suivi de V+ (respectivement

Exemple : Pour 1000 nombres, cela doit ressembler à:
Pair: 518V+482F
SuperieurA(25): 513V+487F