TP7

Partie cours

Dictionnaire

1. Définir un dictionnaire **recetteCrepe** contenant

```
recetteCrepe = {"oeufs" = 3, "farine" = 0.25, "lait" = 0.5} ;
```

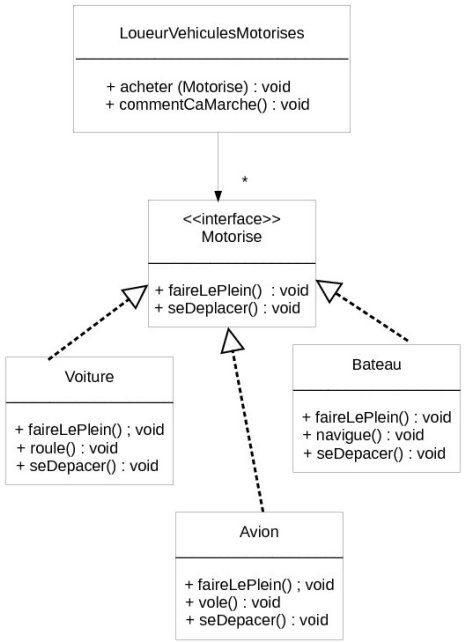
2. puis le dictionnaire

```
magasin = {"oeufs" = 10, "farine"  = 1.15, "lait"  = 5, "sucre" = 4} ;
```

3. Définir une méthode statique **recettePossible** prenant en entrée une recette et un magasin et déterminant si la recette est possible avec les ingrédients disponibles dans le magasin (algorithme déjà fait en python). N'hésitez pas à rechercher les méthodes de *HashMap* de l'API Java.

Interfaces

Implémenter le diagramme de classes UML ci-dessous.



Vous n'écrirez que des affichages dans les codes.

Modèles de classes

On vous demande d'implémenter une classe générique **Simple** possédant un seul attribut, deux constructeurs (l'un sans paramètre, l'autre avec un paramètre du type de l'attribut), un getter et un setter. Vous testerez cette classe dans un programme principal avec un **Integer** (12), un **String** `("toto")` et un **Double** (5.5).

Algorithmique - Dichotomie

Pour rappel, l'algorithme de recherche dichotomique s'écrit par exemple en python :

```
def recherche_dichotomique(liste,elem):
    bas = 0
    haut = len(liste)-1
    while(bas<haut):
        milieu = (bas+haut)//2
        if(liste[milieu]<elem):
            bas = milieu+1
        else:
            haut = milieu
    return bas<len(liste) and elem == liste[bas]
```

Écrivez un code de recherche dichotomique en java. Votre code devra permettre de fonctionner avec l'exécutable suivant :

```java
import java.util.ArrayList;
public class ExecutableDichotomie{

    public static void main(String[] args){
        ArrayList<Integer> liste = new ArrayList<Integer>();
        System.out.println(LibEntiers.recherche(liste,0)); // false
        for(int i=0;i<100000;i++)
            liste.add(2*i);
        System.out.println(LibEntiers.recherche(liste,100)); // true
        System.out.println(LibEntiers.recherche(liste,0)); // true
        System.out.println(LibEntiers.recherche(liste,3)); // false
        System.out.println(LibEntiers.recherche(liste,-3)); // false
        System.out.println(LibEntiers.recherche(liste,400000)); // false

    }

}
```

Attention, ici on testera votre code sur de *grandes* listes (une recherche séquentielle ne fonctionnera pas..).

```
1 Votre classe..
```

Envoyer

Dichotomie liste décroissante

Dans l'exercice précédent on a supposé que la liste est triée par ordre croissant. Supposons maintenant qu'elle est triée par ordre décroissant. Écrivez une méthode statique `rechercheDecroissant` permettant d'exécuter le code suivant :

```java
import java.util.ArrayList;
public class ExecutableDichotomieDecroissant{

    public static void main(String[] args){
        ArrayList<Integer> liste = new ArrayList<Integer>();
        System.out.println(LibEntiers.rechercheDecroissant(liste,0)); // false
        for(int i=100000;i>=0;i--)
            liste.add(2*i);
        System.out.println(LibEntiers.rechercheDecroissant(liste,100)); // true
        System.out.println(LibEntiers.rechercheDecroissant(liste,0)); // true
        System.out.println(LibEntiers.rechercheDecroissant(liste,3)); // false
        System.out.println(LibEntiers.rechercheDecroissant(liste,-3)); // false
        System.out.println(LibEntiers.rechercheDecroissant(liste,400000)); // false

    }

}
```

```
1 Votre classe..
```

Envoyer

/