Les ensembles implémentés via une table de *hashing*

Exercices obligatoires

A. Implémentation de l'interface *Ensemble* via une table de hashing

A1 Implémentez l'interface *Ensemble* en utilisant un tableau de listes.
Appelez cette classe *EnsembleTableHashing*.
La classe *TestEnsembleTableHashing* permet de tester cette classe.
Si c'est nécessaire, pour débugger, appelez la méthode `toString()` donnée. Celle-ci permet d'afficher toutes les listes de la table, même les vides.
Utilisez la classe *ListeSimpleImp* que vous avez écrite ou la solution proposée sur moodle.

A2 Sur moodle, répondez au questionnaire à choix multiples *EnsembleTableHashing*.

B Applications

B1 Le personnel de la société X qui compte 256 employés a le droit de placer sa voiture dans le parking de la société.
Pour éviter les indésirables, l'entrée du parking est munie d'une barrière.
La société décide d'équiper la barrière d'un détecteur de plaques. Celle-ci ne s'ouvrira que pour les plaques autorisées.
Complétez la classe *EnsembleVoituresAutorisees*.
Cette application *possède* les fonctionnalités suivantes :

```
boolean retirerVoiture(Voiture voiture)
boolean ajouterVoiture(Voiture voiture)
boolean voitureAcceptee(Voiture voiture)
```

L'ensemble implémenté via une table de *hashing* est la structure de données la plus adéquate pour conserver les voitures autorisées.
Si la méthode de *hashing* est bonne, ces 3 méthodes sont en O(1)!

La classe *Voiture* possède une méthode de *hashing* qui sera testée à l'exercice C3.

Testez la classe avec la classe *TestEnsembleVoituresAutorisees.* Elle doit être en partie complétée.

C Méthode de *hashing* de la classe *Voiture*

C1 Chaque objet de la classe *Voiture* possède une plaque de type :

Voici un méthode `hashCode()` proposée :

```
      public int hashCode(){
            int resultat = 0;
            for(int i = 0 ; i < 7 ; i++){
                  resultat += (int) this.numPlaque.charAt(i);
            }
            return resultat;
      }
}
```

Calculez le résultat que renverra cette méthode de *hashing* avec les plaques :
(Ne passez pas par une classe Java, le calcul est assez simple : (int)'A' → 65 (int)'0'→ 48))

1AAA000

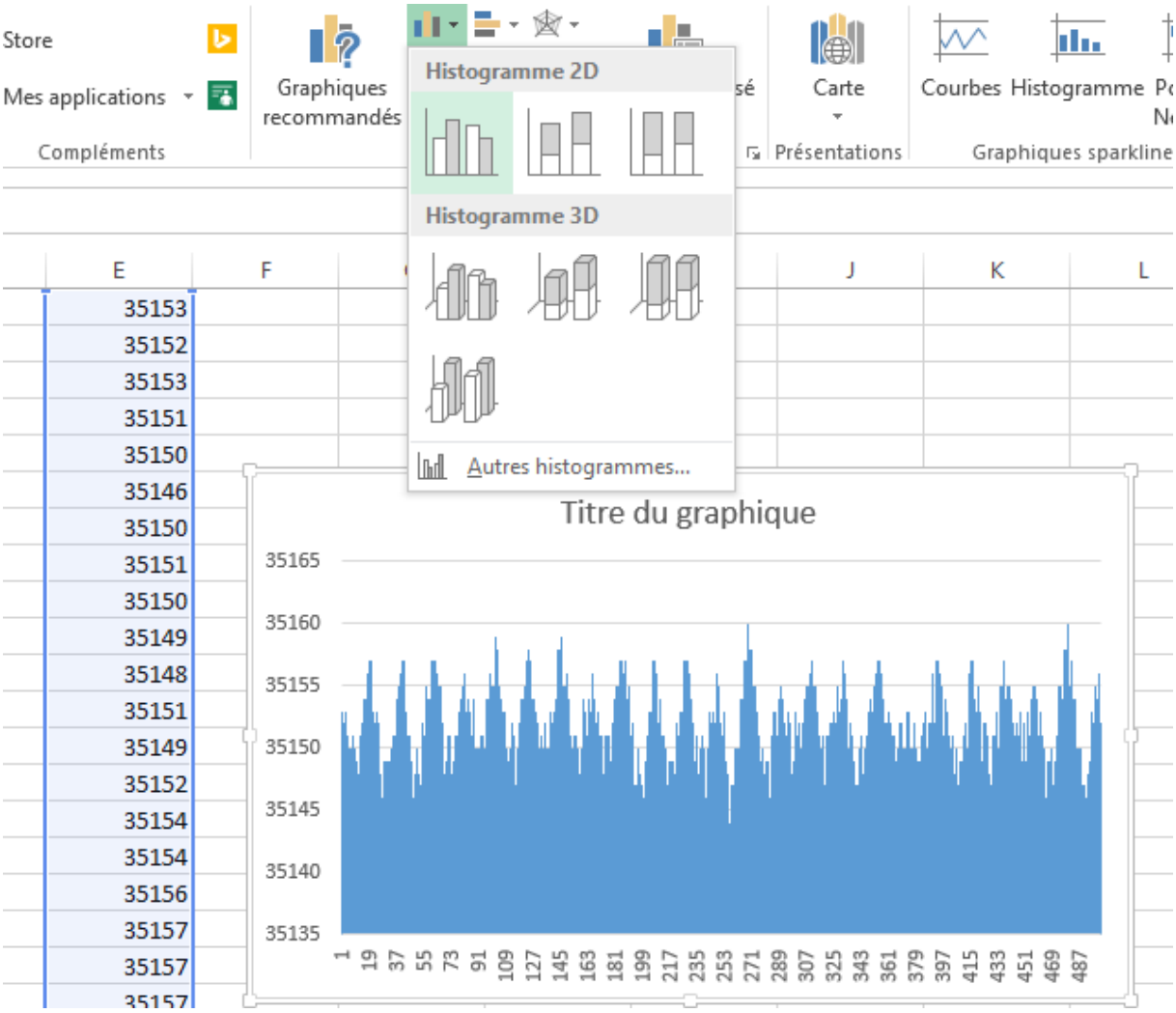
1ZZZ999

Que pensez-vous de cette méthode de *hashing* pour répartir les plaques 1chiffre-3lettres-3chiffres parmi 500 listes ?
(Justifiez votre réponse !)

C2 En vue de réaliser un graphique qui permet de voir comment la méthode de *hashing* de la classe *Voiture* répartit les 17576000 voitures parmi 500 listes, complétez la classe *TestPlaqueDeVoiture*. Pour ce faire, la classe va simuler toutes les voitures portant les numéros de plaque 1chiffre-3lettres-3chiffres. Pour chaque plaque simulée, le programme appelle la méthode de hashing et déduit donc le numéro de la liste où placer la plaque. Après toutes ces simulations, la classe va afficher à l'écran le nombre de plaques obtenue dans chaque liste. Les résultats vont correspondre aux ordonnées du graphique.
Il suffira de reporter (copier-coller) les résultats qui s'affichent à l'écran sur une page Excel et d'insérer un graphique type « Histogramme» avec ces résultats.

Remarque : il n'est pas nécessaire de créer 500 listes, 500 compteurs feront l'affaire ! Réfléchissez ...

Voici la démarche à suivre pour obtenir le graphique sous Excel :



Commencez par sélectionner les données avant de choisir, dans le menu « insertion », le type de graphique histogramme 

C3 Comparez à l'aide de graphiques (en utilisant la classe *TestPlaqueDeVoiture)* différentes méthodes de *hashing* de la classe *Voiture* :

Vous compléterez le document word *graphiques* qui se trouve sur moodle.
On vous demande de soumettre ce document !

Remarque importante :

La classe Object propose une méthode `hashCode()`.
C'est celle-ci que vous redéfinissez ! Mais que se passe-t-il si vous ne le faites pas ?

Voici un exemple :

```java
public class TestHashObject {

    public static void main (String args[]) {
        Voiture voiture1 = new Voiture("1AAA000", "p");
        Voiture voiture2 = new Voiture("1AAA000", "p");
        System.out.println(voiture1.hashCode());
        System.out.println(voiture2.hashCode());
    }

}
```

Console

<terminated> TestHashObject [Java Application] C:\Program Files (x86)\Java\jre1.8.0

```
27134973
1284693
```

Si la méthode `hashCode()` n'est pas redéfinie, elle renvoie l'adresse de l'objet. Dans l'exemple, nous avons 2 instances donc 2 adresses différentes.

a) La classe String a redéfini la méthode `hashcode()`.

La méthode ne renvoie pas l'adresse. Et heureusement l'entier renvoyé est le même pour 2 mêmes chaînes de caractères.
Nous pouvons peut-être l'utiliser ?
C'est-ce qui est fait dans la méthode `hashCode()` proposée par défaut.
Testez-là !

b) Testez la méthode proposée en C1.

c) IntelliJ propose de générer automatiquement les méthodes `hashCode()` et `equals()`.
Faites-le !
Mettez bien en commentaires les méthodes `hashCode()` et `equals()` existantes.
Observez les méthodes générées.
Mise en garde :
C'est bien pratique, mais il ne faut surtout pas oublier de le faire !!!

d) Proposez une méthode de *hashing* « idéale » pour les plaques 1 - 3 lettres – 3 chiffres en vue de répartir les plaques parmi 500 listes.
(N'en essayez pas plusieurs, réfléchissez plutôt à la *String* manipulée !)

C4 Testez votre méthode idéale si on décide de répartir les voitures parmi 350 listes, 600 listes et 2000 listes.
Dans la classe *TestPlaqueDeVoiture*, modifiez la valeur de la constante `NOMBRE_LISTES`.
Revoyez vos conclusions en observant les graphiques obtenus en utilisant votre méthode idéale avec ces nouvelles répartitions.

Exercice défi

A3 Ajouter un attribut `loadFactor` et le constructeur `EnsembleTableHashing(int capacite, double loadFactor)` à votre classe *EnsembleTableHashing*.
Revoyez la méthode `ajouter()`.
Prévoyez un agrandissement de table dans le cas où
taille/capacite > `loadFactor`.
Si votre méthode est au point, voici ce que devrait afficher la classe *TestDefiEnsembleTableHashing :*

```
**************************
Programme Test pour le defi
**************************

table0
table1 5
table2
table3

table0
table1 5
table2
table3 7

table0
table1 9 5
table2
table3 7

table0
table1 1 9
table2
table3
table4
table5 5
table6
table7 7

table0 8
table1 1 9
table2
table3
table4
table5 5
table6
table7 7
```

Exercice supplémentaire

A4 Modifiez la méthode `toString()` de la classe *EnsembleTableHashing*.
Celle-ci doit renvoyer uniquement les éléments de l'ensemble sans faire apparaître la structure de données choisie pour l'implémentation.