TP 6-7
Tables de correspondances : implémentations

1 Contenu et objectifs des TPs

1.1 Objectif général :

Dans ce TP, nous allons étudier une structure de donnée importante, **la table de correspondance**. Celle-ci permet d'associer une donnée d'un type `C` (la *clé*) à une donnée d'un type `V` (la *valeur*). Elle permet, en n'ayant que la clé, de retrouver la valeur.

Voici deux exemples de l'utilité de cette structure :

- Associer des numéros de carte d'étudiant à des noms d'étudiants (C = Integer, V = String) :
  $<$(3459, « Jean Dupont »), (8491, « Marie Martin »),...$>$
- Associer des numéros d'IP à des infos d'utilisateurs (C = String, V = String) :
  $<$(« 238.124.22.53 », « Thomas Dubois, 11 rue des graviers, Rennes »), (« 74.125.136.99 », « Google, 1600 amphiteatre parkway, MountainView »),...$>$

1.2 Rendu attendu :

Ce TP est un TP évalué, à réaliser en binôme, le dépôt se fait sur Moodle.

Pour la remise du TP sur Moodle, **votre binôme doit être enregistré** pour pouvoir faire le dépôt commun. **Si vous avez changé de binôme** entre le TP Sudoku et ce TP, faites en part à votre chargé de TP dès la 1ère séance sur ce sujet de TP.
Vous trouverez l'activité de dépôt dans la section **Rendu de TP** de Moodle :

Date de remise du projet : **11/12/2022 à 23h59 max**.

Tout copier/coller d'Internet ou du TP de vos camarades sera égal à 0 au TP, et cela pour tous les participants à la fraude.

2 Modélisation JAVA

Dans ce TP, on s'intéressera à des **tables de correspondance simples** (aussi appelées dictionnaires) dans lesquelles l'association est de type `String-String`.

Avant de commencer ce TP, vous créerez un répertoire de travail et y copierez le fichier `Dico.java` disponible sur Moodle.
Vous trouverez dans le fichier, la classe abstraite (c.a.d. que vous ne pouvez pas créer d'instance de ce type ; vous ne pouvez que la sous-classer) `Dico` suivante :

```
public abstract class Dico {

        /**
         * Crée une association entre une clé et une valeur.
         * Si une association (cle, autre valeur) existe déjà, elle est écrasée.
         *
         * @param cle Clé de l'association, supposée non nulle
         * @param valeur  Valeur de l'association, supposée non nulle
         */
        public abstract void associe(String cle, String valeur) ;

        /**
         * Supprime l'association (cle, valeur) si elle existe, sinon ne fait rien
         *
         * @param cle Clé à supprimer
         */
        public abstract void supprime(String cle) ;

        /**
         * Renvoie la valeur associée  à la clé indiquée.
         *
         * @param cle Clé à rechercher
         * @return Valeur associée à cle, null sinon
         */
        public abstract String get(String cle) ;

}
```

3 Implémentation d'une table de correspondance sous forme d'une liste

Exercice 1. Faire une liste
Écrivez une sous-classe `DicoListe` de `Dico`. `DicoListe` doit implémenter toutes les méthodes de `Dico` en exploitant **une liste chaînée**. Vous devez implémenter vous-même cette liste, **interdiction d'utiliser les collections Java**.

Remarque : Vous pourrez, pour vous aider, créer une classe publique `Paire` ou exploiter la classe correspondante fournie par Java (à vous de la trouver). **Attention**, la classe `Paire` vous servira aussi pour `DicoArbre` (question suivante) : ne la spécialisez donc pas pour les listes chaînées !

Exercice 2. Tout code mérite son test !
Fournissez des jeux de test pour les différentes méthodes de `DicoListe`.

4 Implémentation d'une table de correspondance sous forme d'un arbre

Exercice 3. Faire un arbre binaire de recherche
Écrivez maintenant une sous-classe `DicoArbre` de `Dico`. `DicoArbre` doit implémenter toutes les méthodes de `Dico` en utilisant un arbre binaire de recherche (vu en cours) sur les clés.

Vous devez implémenter vous-même l'arbre binaire de recherche, et **ne pas recourir aux collections standard Java**.

NB : la suppression vous demandera un peu de réflexion. La page wikipedia sur les arbres binaires de recherche vous donnera la méthode à suivre, à vous de l'implémenter.

Exercice 4. Ce code aussi veut être testé
Fournissez des jeux de test pour les différentes méthodes de `DicoArbre`.

5 Comparaison de performances

Pour vérifier ces différentes implémentations d'une table de correspondance, vous allez conduire des tests de performance. Pour ce faire, vous allez comparer le temps d'exécution des instructions d'insertion, et d'accès aux données dans les deux formats de table.

Recommandations :
- Pour mesurer le temps, utilisez la méthode de classe `nanoTime` de la classe `System`, voir la documentation ici.
- Pour générer un grand nombre de clés et de valeurs, voir la méthode de classe `random` de la classe `Math`, dont la documentation est accessible ici. Celle-ci vous permettra de générer des nombres aléatoires, que vous convertirez en `String` pour les utiliser en clé et valeurs.

Exercice 5. Coder les tests
Créez une classe de test `TestPerformance`, dans laquelle vous insérerez une ou plusieurs méthodes permettant de comparer les performances de vos implémentations. Vous expliciterez en commentaire de la ou des méthodes créée(s) le rôle de celle(s)-ci. Les comparaisons attendues sont les suivantes :

1. Mesurez le temps de 100 000 d'insertions. Comme chaînes pour vos clés-valeurs, prenez des couples de nombres aléatoires compris entre 1 et 100 000, que vous convertirez en `String`.
2. Mesurez ensuite le temps moyen d'accès (= `get`) pour 100 000 instructions d'accès à des clés représentant des valeurs aléatoires entre 1 et 100 000.

Exercice 6. Analyser les résultats des tests
Dans un petit rapport, vous reporterez les valeurs obtenus sur ces tests et discuterez les résultats.

Exercice 7. Vérifier les tendances
Les expériences précédentes utilisent N = 100 000 d'accès/insertions. Il serait intéressant d'observer les performances pour d'autres valeurs de N.

1. Modifiez votre code, afin de pouvoir réaliser facilement ces tests avec d'autres valeurs de N.
2. Dessinez vos résultats dans des courbes avec les valeurs de N en abscisse et les valeurs de temps (temps d'insertion, temps moyen d'accès) en ordonnée.
3. Insérez ces courbes dans le rapport en les commentant.