TP 2

Le TP sera réalisé en Java.

0. Création du projet TP2

Pour créer le nouveau projet, nous allons utiliser l'outil d'importation d'Eclipse :

- récupérez le fichier S7-TP2.tar.gz et sauvez-le sur votre compte
- allez dans **File → Import → General → Existing Projects into Workspace**
- sélectionnez "select archive file" et cliquez sur "Browse"
- sélectionnez le fichier `S7-TP2.tar.gz`

Un nouveau projet `S7-TP2` doit apparaître dans votre espace de travail. Il comprend 3 classes : `principal`, `LecteurDeTuples` et `Tuple` telles qu'elles devaient apparaître à la fin du TP1.

Exécutez le projet et vérifiez que tout marche bien.

1. Classe Client

Créer une classe `Client` contenant les attributs suivants :

- - **nom : String**
- - **prenom : String**
- - **codePostal : String**
- - **pays : String**

remarque : tous les attributs sont de type `String`.

Définissez le constructeur, la méthode `toString`, les getters et les setters, et testez la classe dans le programme principal.

2. Liste de clients

Le but est de générer une liste de clients à partir des données du fichier `ventes.csv` vu dans le TP1. Dans le programme principal qui vous est donné, les tuples sont lus à l'aide de l'objet `h` ("lecteur de tuples") qui "charge" successivement chaque tuple du fichier et l'affiche.

- a - Modifiez la boucle afin que seules les données qui concernent les clients s'affichent (à savoir les champs `NOM_CONTACT`, `PRENOM_CONTACT`, `CODE_POSTAL` et `PAYS`, les autres informations ne doivent plus s'afficher)

NB : Pour récupérer le tuple courant, utilisez la methode `getTuple`:

```
Tuple t = h.getTuple();
```

puis `t.getSchema()` donne le tableau des attributs et `t.getValeurs()` donne le tableau des

valeurs. Il faut donc trouver dans quelle(s) case(s) se trouvent les informations recherchées et les afficher.

- b - Définissez dans la classe `Client` un nouveau constructeur qui prend comme seul paramètre le "lecteur de tuples" h. Le constructeur récupère le tuple courant, et initialise le client avec les champs `NOM_CONTACT`, `PRENOM_CONTACT`, `CODE_POSTAL` et `PAYS` du tuple.

Modifiez le programme principal en créant à chaque tour de boucle un nouveau client c à partir du lecteur de tuples h et en affichant ce client avec:

```
Client c = new Client(h);
System.out.println(c);
```

- c - Nous souhaitons maintenant stocker tous les clients dans une liste.

Dans le programme principal, créez une liste de clients vide :

```
ArrayList<Client> clients = new ArrayList<Client>();
```

puis remplissez la liste en ajoutant un nouveau client à chaque tour de boucle. (n'oubliez pas la commande d'importation : `import java.util.ArrayList;`)

Affichez cette liste de clients. Affichez le nombre d'éléments.

remarque : vous pouvez utiliser la syntaxe simplifiée :

```
for (Client c : clients) {
    system.out.println(c);
}
```

Un examen attentif de cette liste permet de constater que les certains clients y sont présents plusieurs fois. Essayez de répérer ces clients présents plusieurs fois.

3. Comparaison de clients

Pour obtenir une liste "sans doublons", il est nécessaire de supprimer les clients présents plusieurs fois.

Nous avons pour cela besoin d'un opérateur permettant de comparer deux clients. Il existe l'opérateur == qui :

- compare les valeurs pour les objets simples (entiers, chaînes de caractère),
- compare les étiquettes pour les objets structurés

Ainsi `a == b` renvoie `true` si a et b sont deux étiquettes pour un même objet en mémoire, et `false` `sinon`. Ceci n'est pas adapté pour comparer deux clients.

Ajoutez donc dans la classe `Client` une méthode `egaleClient` qui compare un par un les attributs du client courant et du client fourni en paramètre:

- + **egaleClient(Client) : Boolean**

Cette méthode doit retourner un booléen.

Testez cette méthode dans votre programme principal:

```
Client a = new Client("Caccialotti","Michel","13190","France");
Client b = new Client("Caccialotti","Michel","13190","France");
Client c = new Client("Caccialotti","Lionel","13380","France");
System.out.println(a.egaleClient(b)); // doit afficher true
System.out.println(a.egaleClient(c)); // doit afficher false
```

Remarque : dans Eclipse, cette méthode peut être générée automatiquement. Allez dans la classe `Client` et faites: **source → generate hashCode() and equals()** qui génère deux méthodes complémentaires `hashCode` et `equals`.

La méthode `equals` est équivalente à la méthode `egaleClient`. Comparez votre code avec le code généré par Eclipse. Vérifiez que le comportement est le même:

```
    System.out.println(a.equals(b)); // doit afficher true
    System.out.println(a.equals(c)); // doit afficher false
```

La méthode `hashCode` sert à attribuer un nombre entier à chaque objet créé. Il s'agit d'un identifiant construit à l'aide d'une fonction pseudo-aléatoire. Le hashcode est construit de manière à ce que deux objets similaires aient le même hashcode et que le nombre de hashcodes possibles soit suffisamment élevé pour que les objets différents aient la plupart du temps des hashcodes différents.

Affichez les hashCodes des trois clients `a`, `b` et `c`.

Vérifiez que :

```
    System.out.println(a.hashCode()==b.hashCode()); // doit afficher true
    System.out.println(a.hashCode()==c.hashCode()); // doit afficher false
```

4. Ensemble de clients

Nous allons maintenant générer un Ensemble de clients. Contrairement à la Liste, il ne peut exister deux éléments identiques au sein d'un même ensemble. Nous utilisons pour cette question une nouvelle structure de données appelée le `HashSet` :

```
import java.util.HashSet;
```

Remarque : l'utilisation de l'ensemble `hashSet` nécessite que les méthodes `hashCode` et `equals` soient définies dans la classe correspondante, ce qui a été fait dans la partie 3.

La commande suivante crée un ensemble `ensClients` vide:

```
HashSet<Client> ensClients = new HashSet<Client>();
```

Pour ajouter les clients `a`, `b` et `c` à l'ensemble :

```
ensClients.add(a);
ensClients.add(b);
ensClients.add(c);
```

Affichez le nombre de clients avec :

```
System.out.println(ensClients.size());
```

Pourquoi `ensClients` ne contient-il que 2 clients après ces trois opérations?

Supprimons maintenant des clients de l'ensemble :

```
ensClients.remove(a);
ensClients.remove(c);
```

Vérifions que l'ensemble est bien vide :

```
System.out.println(ensClients.isEmpty()); // doit afficher true
```

Maintenant, il ne reste plus qu'à remplir `ensClients` avec les clients du fichier. Reprenez la question 2 en remplaçant l'`ArrayList` par un `HashSet`. Combien de clients obtenez-vous? Affichez la liste des clients.

Remarque : pour afficher le contenu de l'ensemble :

```
for(Client c : ensClients){
    system.out.println(c);
}
```

Question optionnelle :

Créez une classe **EnsembleDeClients** contenant les méthodes suivantes:

- **EnsembleDeClients()** crée un ensemble vide
- **addClient(Client) : void** ajoute un client à l'ensemble
- **getClientByName (String) : Client**
- **contains(Client) : boolean**

et testez-la dans le programme principal.

5. Statistiques sur les clients

Nous souhaitons maintenant effectuer des statistiques élémentaires sur notre ensemble de clients. Nous allons essayer d'estimer la répartition géographique de nos clients en comptant le nombre de clients par pays.

La table associative `mapClients` servira pour compter le nombre de clients par pays : à chaque pays (de type String) sera associé le nombre de clients correspondant. Nous utiliserons pour cela une structure de type `Hashtable`:

```
import java.util.HashSet;
import java.util.Enumeration;
...
Hashtable<String,Integer> mapClients = new Hashtable<String,Integer>();
```

- pour lire le contenu d'une case de la table associative, il faut définir la clé puis lire la valeur à l'aide de `mapClients.get(cle)`
- pour remplir une case de la table associative, il faut définir la clé et la valeur, puis mettre à jour la table avec `mapClients.put(cle,valeur)`

Ici la clé est de type texte (`String`) et la valeur de type entier (`int`).

Pour remplir la table associative, vous utiliserez l'algorithme suivant:

```
Pour tout les clients c de l'ensemble :
    clé ← c.getPays()
    si mapClients contient la clé :                 //(containsKey)
         valeur ← mapClients.get(cle) +1
    sinon
         valeur ←  1
    mettre à jour la table avec le couple (clé, valeur)
```

Une fois la table remplie, affichez le nombre de clients par pays (vous devrez pour cela définir un parcours sur des objets de type "Entry").

```
for (Entry<String, Integer> e : mapClients.entrySet()) {
    String pays = e.getKey();
    int valeur = e.getValue();
    // traitements
}
```