TD N°1

Bonne formation des objets - Instanciation - Envoi de messages

1 Compilation en Java

Ce petit exercice a été écrit en Java et permet de construire une liste de contacts.
Les sources du programme sont accessibles depuis l'archive `td1.tgz`
Les classes suivantes ont été construites :

— `Main` c'est la classe qui permet de définir l'objet principal du programme
— `ContactSet` : la liste des contacts
— `Contact` : le contact identifié par un nom, une adresse, etc.
— `Mail`, `PhoneNumber`, `Town` : adresse électronique, numéro de téléphone et Ville
— `RandomAddress` : une classe pour générer des adresses aléatoires

1. Compiler le programme en utilisant la commande en ligne **javac**
2. Exécuter le programme en utilisant la commande en ligne **java**
3. Compiler et Exécuter le programme en utilisant la commande gradle `gradle run`

2 Code

Ouvrir le code `app/src/main` avec l'éditeur de code de votre choix, l'éditer et modifier si nécessaire.

Création d'instances - Encapsulation - Responsabilité

On s'interrogera sur la pertinence de la création d'instances. Quel objet est supposé construire une autre instance, c'est-à-dire un autre objet ? Par exemple, est-il légitime qu'un contact construise une instance de Town alors que cette instance pourrait être partagée par d'autres contacts ? On examinera systématiquement le code pour que le programme soit cohérent de ce point de vue.

On observera et on corrigera si nécessaire que chaque objet est correctement encapsulé et responsable de ses propres données.

On observera et on corrigera si nécessaire l'ordre entre les objets des envois de messages.

Objets bien formés

Chaque objet est responsable des données qu'il contient. Nous allons modifier le code de sorte qu'il n'est pas possible de construire un objet `Contact` qui ne soit pas bien formé. Savoir :

— Ses nom et prénom doivent respecter une norme typographique (pas plus de 64 caractères alphabétiques)
— Son email doit être bien formé
— Son téléphone doit correspondre à une numérotation française
— Le zipcode doit correspondre à la ville

Astuces :

— On pourra utiliser les expressions régulières
— On pourra utiliser la constante `towns` de la classe `Town` en la parcourant (ce qui n'est pas idéal, mais suffisant pour l'instant)
— En cas d'échec, plutôt de de quitter brutallement le programme, on pourra créer une exception

Instances uniques

L'instance de l'objet de type `Town` qui correspond à *Marseille - 13000* mérite d'être seule dans le programme pour éviter les doublons. On utilisera le mot clef **static** pour rendre unique les instances de **Town**

Deux contacts différents doivent avoir des identifiants différents. Ils peuvent cependant être homonymes, mais ne peuvent pas avoir les mêmes contacts email. Modifier le programme pour forcer cela.

3 Bonne formation des ensembles

L'implémentation d'un ensemble est une liste telle que deux élements ne sont pas égaux entre eux. L'égalité des éléments est définie dans la class `HashSet<T>` par les méthodes `hashCode` et `equals`. Ces deux méthodes sont implémentées dans la classe `java.lang.Object`.

1. Expliquez pourquoi `hashCode` **et** `equals` doivent être implémentées dans T pour utiliser HashSet<T>
2. Vérifier que deux contacts égaux peuvent être ajoutés dans la liste des contacts
3. Corrigez cela en redéfinissant `hashCode` et `equals`

4 Bonus

1. Expliquez pourquoi `T` doit-il implémenter l'interface `Comparable<T>` pour utiliser `TreeSet<T>`
2. Modifier le code pour utiliser `TreeSet<Contact>` à la place de `HashSet<Contact>` dans `ContactSet`
3. Dire les avantages et inconvénient de cette modification. Est-elle souhaitalbe finalement ?

5 Suite

Il n'est pas très élégant de lister d'une façon ou d'une autre la liste des communes françaises et leur code postal dans la classe `Town`.

Nous allons changer cela pour que l'information soit automatiquement traitée depuis un fichier de données CSV que l'on pourra charger depuis `https://www.data.gouv.fr/fr/datasets/base-officielle-des-codes-postaux/`

1. Télécharger le fichier et produire un fichier towns.csv contenant deux colonnes pour les codes postaux et les noms de commune.

   Les commandes de manipulation de texte **cut**, **paste**, **sort**, **grep**, etc., peuvent être utiles pour cette transformation. Si le processus devient trop complexe, il est recommandé d'utiliser des outils comme perl, python ou tout autres langages de programmation adapté au traitement de texte.

   exemple :

```
iconv -f ISO-8859-1 -t UTF-8 < ~/Downloads/019HexaSmal.csv
    | sort -u | cut -d';' -f3,2
    | awk -F ';' '{temp = $1; $1 = $2; $2 = temp; OFS=";"; print}'
    | tr '[:lower:]' '[:upper:]' > towns.csv
```

2. Dans la classe Town, créer un attribut statique towns permettant d'associer un code postal à un nom de commune. Pour cela, on utilisera l'implémentation HashMap<K, V> de Map<K, V>.

   (a) Expliquer pourquoi cet attribut est statique.
   (b) Que faut-il comme type d'objet pour représenter le code postal. Est-il utile d'implémenter equals et hashCode dans la cas que vous choisissez ?

3. Toujours dans la classe Town, créer une méthode pour charger le fichier et de remplir l'application towns.
4. Ajouter une vérification dans le constructeur pour tester l'existence du nom de la ville
5. Ajouter une méthode pour avoir le code postal d'une ville à partir de son nom et inversement
6. Est-il souhaitable que chaque ville conserve son code postal et son nom ? Qu'est-il le plus profitable de faire ?