

Interface Repertoire et codage d'un modèle

Après de longues heures d'analyse, le choix s'est porté sur la modélisation suivante :

Repertoire.java

```
import java.util.List;

public interface Repertoire{

    public static final int PAS_DE_TRI=0;
    public static final int TRI_PAR_ORDRE_ALPHABETIQUE=1;
    public static final int TRI_PAR_NOMBRE_DE_NUMERO=2;

    /**
     * permet d'ajouter un contact dont on renseigne le nom et le numéro de téléphone
     */
    public void ajouteContact(String nom, String numero);

    /**
     * Cette méthode permet de supprimer totalement un contact
     * Elle propage une exception si le nom passé en paramètre
     * ne correspond à aucun contact du répertoire
     */
    public void supprimeContact(String nom) throws PasDeContactException;

    /**
     * renvoie la liste des noms de tous les contacts du répertoire
     */
    public List<String> getNoms();

    /**
     * renvoie la liste triée des noms de tous les contacts du répertoire
     * La liste est triée en fonction de la valeur du paramètre typeDeTri
     * PAS_DE_TRI -> pas de tri spécifique
     * TRI_PAR_ORDRE_ALPHABETIQUE -> les noms sont triés par ordre alphabétique
     * TRI_PAR_NOMBRE_DE_NUMERO -> le nom des contacts sont triés par ordre
     * décroissant de leur nombre de numéros de téléphone
     */
    public List<String> getNoms(int typeDeTri);

    /**
     * renvoie la liste de tous les numéros de téléphone du contact dont
     * le nom est passé en paramètre
     */
    public List<String> getNumeros(String nom);

    /**
     * renvoie la liste des noms de tous les contacts associés
     * au numéro de téléphone passé en paramètre
     * Cette méthode propage une exception si le nom passé en paramètre
     * ne correspond à aucun contact du répertoire
     */
    public List<String> rechercheNumero(String numero) throws PasDeContactException;

    /**
     * permet d'initialiser le répertoire avec quelques contacts de sorte
     * à avoir un jeu d'essai non vide
     */
    public void initRepertoire();
}
```

L'objectif de ce TP est d'écrire au moins une implémentation de l'interface `Repertoire`.

Première implémentation : la classe RepertoireMap

Pour cette première implémentation, on va modéliser les données à l'aide d'un dictionnaire dont les clés sont les noms des contacts, et les valeurs une liste de numéros de téléphones (chaines de caractères)

Ainsi, notre classe aura un attribut de type

```
Map<String, List<String>>.
```

Ranger les fichiers d'un projet java

Il est temps pour vous de savoir ranger à la main les fichiers de vos projets java.

On crée un dossier par projet. Ici par exemple le dossier `Repertoire`. Dans ce dossier, on crée trois sous-dossiers :






1. Créez le fichier `Repertoire.java` et écrivez le **code minimal** d'une classe `RepertoireMap` qui implémente l'interface `Repertoire`. Dans un premier temps, on écrit un minimum de code de façon à ce que "ça compile". A ce stade, vos méthodes sont donc vides, ou presque.

Il est très probable que vous soyez obligé d'écrire d'autres classes ;-)

2. Créez un exécutable qui vous permettra de tester les méthodes de votre classe `RepertoireMap`
3. Codez les méthodes une à une en prenant soin de les tester dans votre exécutable.

Connexion avec l'interface graphique

4. Une fois que votre code compile et que vos méthodes sont implémentées et testées, téléchargez la *vue* développée par une autre équipe de codeurs ainsi que les *controleurs* :

-  la *vue* ;
-  le *controleur* qui permet d'ajouter un contact ;
-  le *controleur* qui permet d'effectuer une recherche ;
-  le *controleur* qui permet de supprimer un contact ;
-  le *controleur* qui permet de trier.

Compilez et exécutez la classe `VueRepertoire` et essayez d'utiliser l'application.

5. Quelles améliorations pourraient être apportées à cette application ?
6. Implémentez une ou deux améliorations proposées

Facultatif

7. Proposez une autre implémentation de `Repertoire`

- un dossier `src` pour les fichiers `.java`
- un dossier `bin` ou `out` pour les fichiers `.class`
- un dossier `doc` dans lequel on générera la javadoc

Les commandes qui suivent supposent que vous vous trouvez à la racine du projet (i.e. dans le dossier `Repertoire` dans notre exemple)

Pour **compiler** (les fichiers *.java sont dans src/ et on veut mettre les fichiers .class dans bin/)

```
javac -d bin src/Executable.java
```

Pour **exécuter** le fichier Executable (les .class étant dans bin)

```
java -cp bin Executable
```

ou encore

```
java -classpath bin Executable
```

Pour **générer la javadoc** (dans le répertoire doc/)

```
javadoc -d doc src/*.java
```

ou encore

```
javadoc -d doc charset utf8 -private -noqualifier java.la
```

Détail des options proposées :

- `charset utf8` permet de préciser l'encodage utf8
- `-private` pour faire figurer les attributs et méthodes privées (private)
- `-noqualifier java.lang` pour ne pas faire figurer le chemin java.lang devant String, Boolean ... etc

Ouvrez ensuite le fichier `index.html`. C'est magique !!