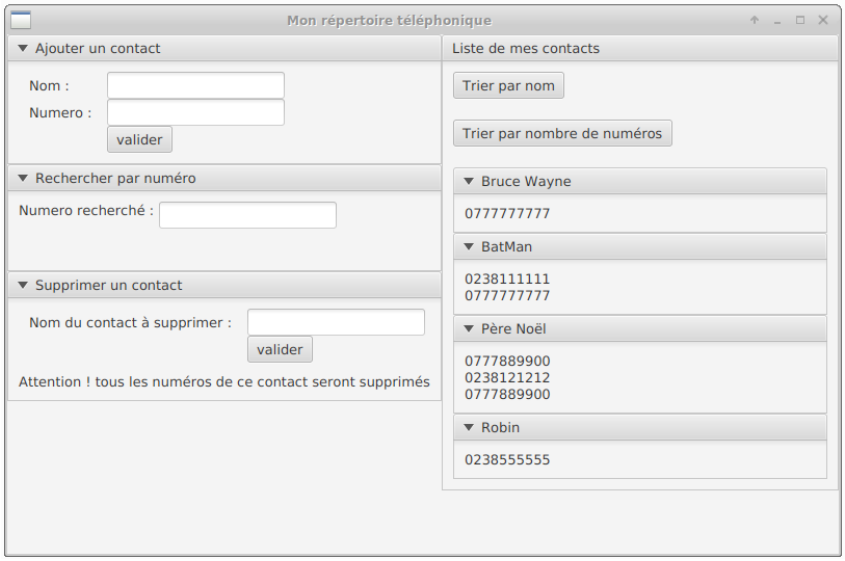
2. Feuille de TP semaine 11

L'objectif de ce TD/TP est de réaliser le **modèle** d'une application qui permettra de gérer une liste de contacts téléphoniques.

Voici un exemple de visuel que l'on souhaite :



Gestion d'une liste de contacts téléphoniques - une petite analyse

Cette application doit permettre de gérer une liste de contacts avec au minimum les fonctionnalités suivantes :

- ajouter un nouveau contact
- ajouter un numéro de téléphone à un contact déjà présent dans mon répertoire
- modifier le nom d'un contact
- supprimer un contact
- afficher tous les numéros de téléphone d'un contact
- afficher tous les contacts qui ont un numéro de téléphone donné
- afficher la liste de tous les contacts en triant les noms par ordre alphabétique
- afficher la liste de tous les contacts en les triant par ordre décroissant du nombre de numéros de téléphone.

On se propose d'écrire une classe qui servira de *modèle* à cette application.

1. Quel(s) attribut(s) vous semble(nt) approprié(s) pour décrire les données d'une telle classe.
2. Donnez le profil des méthodes qu'il vous semble nécessaire d'implémenter.

Interface Repertoire et codage d'un modèle

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

Repertoire.java

Ranger les fichiers d'un projet java

Un rappel pour ranger les fichiers et utiliser javaFX.

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

- un dossier `src` pour les fichiers `.java`
- un dossier `bin` ou `out` pour les fichiers `.class`
- un dossier `doc` dans lequel on génèrera 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 --module-path /usr/share/openjfx/lib/ --add-modules javafx.controls -d bin src/*.java javac -d bin src/Executable.java` (à ne pas mettre pour compiler avec javafx faire ci-dessus)

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 --module-path /usr/share/openjfx/lib/ --add-modules javafx.
controls -d doc src/*.java
```

`javadoc -d doc src/*.java` (à ne pas mettre)

ou encore

```
javadoc --module-path /usr/share/openjfx/lib/ --add-modules javafx.control -d doc
charset utf8 -private -noqualifier java.lang src/*.java
```

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 !!

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>>`.

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 executable 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 d'effectuer une modification` ;
- `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`