TP 4 Classes et objets

26 octobre 2017

Dans ce TP nous allons construire progressivement une application en ligne de commande de gestion d'un carnet d'adresse simple, avec ajout de fiche et recherche par numéro de téléphone.

1 La classe des personnes

1. Définir une classe `Person` qui contient trois champs privés de type `String` : `firstname` (prénom), `lastname` (nom), `phone` (numéro de téléphone). [1]
2. Écrire un constructeur `Person(String firstname, String lastname, String phone)` qui remplit les trois champs de l'objet avec ses arguments.
3. Rajouter deux méthodes publiques d'accès aux champs de la classe qui renvoient respectivement l'identifiant et le téléphone de la personne :

```
public String getIdent()      // renvoit la concaténation nom + prénom
public String getPhone()      // renvoit le numéro de téléphone
```

4. Dans la méthode `main` de la classe principale, instancier un objet de la classe `Person`, puis afficher le retour de l'appel à sa fonction `getIdent()`, pour vérifier que tout marche bien.

2 La classe carnet d'adresse

1. Définir une classe `Phonebook` qui contient deux champs :
   (a) un tableau `contacts` d'objets de type `Person`, initialisé à une taille de 100. [2] Notez que la valeur par défaut d'un tableau d'objet est `null`.
   (b) un entier `next`, qui sera la première case vide du tableau. Autrement dit, si le carnet d'adresse contient trois personnes `a`, `b` et `c`, alors `contacts` vaudra `{a, b, c, null, null ... }` (97 fois `null`), et `next` vaudra 3.
2. Ajouter à `Phonebook` une méthode de signature :

   ```
   public void addPerson(String first, String last, String phone)
   ```

   qui instancie un objet `Person` en remplissant ses champs, et l'ajoute à `contacts` à la case d'indice `next` (la première libre). N'oubliez pas d'incrémenter `next`, pour que les prochains ajouts n'écrasent pas les précédents.
3. Ajouter ensuite une méthode :

   ```
   public String searchByPhone(String phone)
   ```

1. On représente les numéros comme des chaîne de caractères car une représentation en `int` ne prendrait pas en compte les zéros devant le nombre (pour Java, 00042 est la même chose que 42).
2. 100 est la taille maximale de notre carnet d'adresse ; on suppose qu'elle est assez grande pour que l'on ne la dépasse jamais.

qui renvoit l'identifiant du contact dont le numéro est `phone`, ou `null` s'il n'y a pas de personne associée à ce numéro.

4. Tester ces dernières en exécutant dans votre `main` le code suivant :

```
PhoneBook c = new PhoneBook();
c.addPerson("Paul", "McCartney", "0123456789");
c.addPerson("John", "Lennon", "0698765432");
System.out.print("Quel numéro cherchez-vous? ");
String s = new Scanner(System.in).nextLine();
System.out.print("Ce numéro appartient à ");
System.out.println(c.searchByPhone(s));
```

3 La classe des *contacts* (personnes ou entreprises)

Jusqu'à maintenant, le carnet d'adresse n'était pas adapté à l'ajout d'entreprise : une entreprise n'a pas de nom et prénom, juste un nom. Nous allons rajouter la capacité d'ajouter un nouveau type d'entrée, les entreprises, avec les champs qui leurs sont adaptés.

1. On définit l'interface `Contact` de cette façon :

```
interface Contact {
  String getPhone();
  String getIdent();
}
```

   que `Person` implémente l'interface `Contact`.

2. Définir une classe `Company` qui implémente `Contact`, similaire à `Person` mais n'ayant que deux champs : `name` et `phone`. Définir les méthodes nécessaires.

3. Modifier votre classe `Phonebook` de façon à stocker dans le tableau des contacts des objets implémentant `Contact`, et non plus que des `Person`. Ajouter alors une méthode :

   ```
   public void addCompany(String name, String phone)
   ```

   qui ajoute une entreprise au carnet d'adresse.

4. Tester ces dernières en rajoutant dans `main` la ligne :

   ```
   c.addCompany("Abbey Road Studios", "0490548133");
   ```

   Remarquez que grâce à l'interface commune `Contact`, vous n'avez pas eu à adapter la fonction de recherche à la présence d'entreprises dans votre carnet d'adresse.