TP n°6– Collections Java

Exercice 1. On souhaite réaliser une application de gestion d'un annuaire téléphonique permettant d'associer à une personne un ou plusieurs numéros de téléphone.

On va donc travailler avec deux classes `Personne` et `NumTel` qui seront fournies en TP et dont la documentation est donnée en annexe.

• **`Personne`**: classe représentant une personne définie par un nom, un prénom et une civilité (M, Mlle, Mme).

• **`NumTel`** : classe représentant un numéro de téléphone, défini par le numéro proprement dit et un code indiquant la nature du numéro (numéro de portable, numéro de poste fixe professionnel, numéro de poste fixe à domicile, numéro de Fax).

1. Créer une classe `ListeNumTel` permettant de gérer une liste de numéros de téléphone.

   Proposer un constructeur sachant qu'une liste doit toujours contenir au moins un numéro de téléphone ainsi que les méthodes suivantes:

   - `boolean ajouter(int index, NumTel num)`

   Ajoute un numéro à une position donnée dans la liste, sans effet si le numéro est déjà présent dans la liste.

   - `boolean ajouterFin(NumTel num)`

   Ajoute un numéro à la fin de la liste, sans effet si le numéro est déjà présent dans la liste.

   - `boolean contientNumero(int num)`

   Teste la présence d'un numéro dans la liste.

   - `java.util.Iterator<NumTel> iterator()`

   Renvoie un itérateur sur les numéros de téléphone contenus dans la liste.

   - `int nbNumeros()`

   Retourne le nombre de numéros de la liste (>=1).

   - `NumTel numero(int index)`

   Retourne le i^{ème} numéro de la liste.

   - `NumTel premierNumero()`

   Retourne le premier numéro de la liste (il existe forcément).

   - `boolean retirer(int num)`

   Enlève un numéro de la liste. Si le numéro existe, retourne `true` sinon `false`. Cette opération n'est possible que si la liste contient au moins deux numéros (`nbNumero()>1`).

   - `java.lang.String toString()`

   Retourne la séquence des numéros contenu dans cette liste.

2. Créer une classe `Annuaire` permettant d'associer à une personne une liste de numéros de téléphone. Chaque personne ne doit se trouver qu'une seule fois dans l'annuaire.

   Cette classe proposera un constructeur sans paramètre et les méthodes suivantes :

- `boolean ajouterEntree(Personne p, ListeNumTel nums)` : ajoute une nouvelle entrée dans l'annuaire.

Si p n'existe pas, on crée une nouvelle association (p,nums) et le booléen `true` est retourné; sinon le booléen `false` est retourné et la méthode est sans effet.

- `ListeNumTel numeros(Personne p)` : retourne la liste des numéros d'une personne. Si la personne est absente retourne `null`.

Quelle méthode doit-on ajouter à la classe Personne pour effectuer cette recherche ? Proposer une implantation.

- `java.util.Iterator <Personne> personne()` : retourne un itérateur sur l'ensemble des personnes contenues dans l'annuaire.
- `void ajouterNumeroFin(Personne p, NumTel n)` : ajoute un numéro à la fin de la liste des numéros d'une personne.

Si la personne n'existe pas, on crée une nouvelle entrée pour cette personne avec comme liste de numéros associée la liste constituée du numéro passé en paramètre.

- `public NumTel premierNumero (Personne p)` : retourne le premier numéro d'une personne si cette personne est présente dans l'annuaire, sinon retourne `null`.
- `public void supprimer (Personne p)` : supprime une personne dans l'annuaire si celle-ci est présente.
- `public void supprimerNumero (int n, Personne p)` : supprime le numéro pour une personne donnée. Si le numéro est le seul numéro de la personne, retire la personne.
- `public Set<Personne> lesEntreesPour (String nom)` : retourne l'ensemble des personnes ayant le nom passé en paramètre.
- `public String toString ()` : retourne une chaîne de caractères décrivant l'ensemble des personnes de l'annuaire. Chaque ligne contient les informations d'une seule personne.

3. On souhaiterait consulter les personnes de l'annuaire dans l'ordre alphabétique.

   Quelles modifications devrait-on apporter et dans quelles classes?

4. Proposer une classe `TestAnnuaire` avec une méthode `main` permettant d'insérer des personnes dans un annuaire, chacune avec une liste de numéros de téléphone associée.

Exercice 2.

1. Construire une classe `Article` caractérisée par deux attributs : `numero` de type `int` et `description` de type `String`. Cette classe devra implémenter l'interface `Comparable`. La comparaison se fera suivant le champ `numero`.
   Cette classe devra également fournir les méthodes héritées de la classe `Object` :
   - `public String toString ()` qui retourne sous forme d'une chaîne de caractères les deux caractéristiques d'un article et
   - `public int hashCode ()` qui pourra être générée automatiquement.

2. Écrire une classe `ArticlesTest` permettant de manipuler un ensemble d'articles. On ajoutera des articles à cet ensemble puis on affichera le contenu de l'ensemble en triant les articles suivant leur numéro.

On ne souhaite pas ajouter deux articles ayant même numéro. Quelle méthode faut-il redéfinir dans la classe `Article` afin d'éliminer les doublons dans l'ensemble ? Redéfinissez-la.

3. Compléter cette classe en proposant une autre méthode de tri des articles basée sur la comparaison de leur description.
   Créer un ensemble trié suivant cette nouvelle méthode à partir de l'autre ensemble et afficher son contenu.

ANNEXE :

Classe Personne

Field Summary

`static int` **`INCONNU`**

`static int` **`MLLE`**

`static int` **`MME`**

`static int` **`MR`**

Constructor Summary

`Personne``(int civilite, java.lang.String nom, java.lang.String prenom)`
crée un personne en indiquant sa civilité (valeurs possibles : Personne.INCONNU, Personne.MLLE, Personne.MME, Personne.MR), son nom et son prénom.

`Personne``(java.lang.String nom, java.lang.String prenom)`
crée un personne en indiquant son nom et son prénom, sa civilité n'est pas spécifiée et est initialisée à INCONNU.

Method Summary

`int` **`getCivilite`**`()`
retourne la civilité de la personne

`java.lang.String` **`getNom`**`()`
retourne le nom de la personne

`java.lang.String` **`getPrenom`**`()`
retourne le prénom de la personne

void **setCivilite**(int civilite)
modifie la civilité de la personne.

java.lang.String **toString**()

Classe NumTel

Représente un numéro de téléphone. Un numéro de téléphone est défini par un couple :

- un entier : le numéro de téléphone proprement dit.
- une lettre qui indique le type de numéro ('T' : poste fixe professionnel, 'D' poste fixe domicile, 'P' : portable, 'F' :fax).

Field Summary

static char **FAX**
fax

static char **FIXE_DOM**
téléphone fixe domicile

static char **FIXE_PROF**
téléphone fixe professionnel

static char **INCONNU**
nature du numéro inconnue

static char **PORTABLE**
téléphone portable

Constructor Summary

NumTel(int num)
crée un numéro de téléphone de type inconnu

NumTel(int num, char type)
crée un numéro de téléphone d'un type donné.

Method Summary

`boolean` **`equals`**`(java.lang.Object o)`
teste l'égalité de ce numéro de téléphone avec un autre.

`int` **`getNumero`**`()`
retourne le numéro de téléphone

`char` **`getType`**`()`
retourne le type de ce numéro de téléphone

`int` **`hashCode`**`()`
redéfinition de la méthode hashCode pour rester cohérent avec la méthode equals. deux NumTel égaux doivent produire le même hashCode.

`void` **`setType`**`(char type)`
change le type de ce numéro de téléphone

`java.lang.String` **`toString`**`()`
renvoie une représentation textuelle du numéro de téléphone sous la forme d'une chaîne de caractères.