TP ANNUAIRE

P.Morat

Objectifs :

Ce Tp permet de mettre en œuvre le principe de polymorphisme. Il utilise la librairie java.util, ce qui permet de découvrir quelques unes de ses classes. Il permet d'aborder au travers d'une application simple les structures de données de l'API JAVA (Map, List, LinkedList, ArrayList, HashMap, TreeMap, Set) et les itérateurs permettant leurs manipulations.

Avant-propos

- Consulter la présentation sur les collections en Java (transparents sur les collections).
- Récupérer les fichiers contenant les éléments fournis.
- Vous construirez votre projet dans un package nommé « jus.aoo.annuaire ».
- **Vous ferez usage de la forme d'itération « foreach » autant que possible.**

1ère partie : Elaboration de classes

On souhaite réaliser des classes servant de base à une application de gestion d'un annuaire téléphonique. On définit pour cela 3 classes :

- Personne : notion de personne définie par un nom, un prénom et une civilité.
- Numeros : liste de numéros de téléphone (un numéro est représenté par un String)
- Annuaire : notion d'annuaire, c'est à dire ensemble d'associations <p,n> avec p∈Personne et $n \in Numeros$.

Pour cette application vous disposez des classes suivantes :

- Util qui contient entre-autres la méthode statique suivante :

```
/**
* retourne une Map contenant les associations Personne-Numero correspondant au fichier texte donné
* chaque ligne du fichier doit être de la forme :
* Civilite Nom Prenom "Numero1" "Numero2" .... <CR> (Civilite = Mr ou Mme ou Melle).
* le fichier doit se terminer par une ligne vide.
* Il doit exister un constructeur dans la classe Personne de signature : Personne(int,String,String),
* tel que le premier paramètre peut avoir les valeurs suivantes : 0=>INCONNU, 1=>MELLE, 2=>MME et 3=>MR,
* les paramètres suivants représentent respectivement le nom et le prénom.
* Il doit exister une classe Numeros admettant un constructeur ayant un numéro en argument
* @param file le nom du fichier d'entrée
* @return la table correspondant à l'annuaire chargé
*/
public static Map<Personne,Numeros> importFromFile(String file)
```

1. Réalisation de la classe Personne

La classe « Personne » dispose de trois champs, pour le codage de la civilité vous utiliserez un type enum qui établira l'extension suivante du type : INCONNU, MELLE, MME, MR. Vous ferez en sorte que l'on dispose de fonctionnalité permettant d'obtenir une valeur de civilité à partir de sa représentation textuelle

ou de son rang, vous proposerez aussi une représentation textuelle plus conforme à la bienséance : Mademoiselle, Madame, Monsieur. Attention cette classe est utilisée dans l'interface qui vous est fournie, **il s'ensuit, qu'en l'absence de spécifications claires, vous devez assurer la conformité de la déclaration aux utilisations qui en sont faites**.

Les méthodes suivantes seront réalisées:

- *constructeur à partir d'un nom et d'un prénom;*
- *constructeur à partir d'une civilité, d'un nom et d'un prénom;*
- *les différents accesseurs nécessaires…*
- *équivalence établie sur la base de nom et du prénom*
- *fonction de hashCode (cf. class Object pour obtenir les spécifications de cette méthode),*
- *fonction toString (exemple: "Monsieur Jean Martin"),*
- *…*

Attention : trop peut être l'ennemi du bien. Chaque fonction proposée doit pouvoir être justifiée.

2. Réalisation de la classe Numeros

On utilisera une « List » comme structure de donnée pour gérer les numéros. Invariant : un objet de type « Numeros » possède au moins un numéro de téléphone.

Méthodes offertes par la classe :

```
/** Constructeur d'une liste à un seul numéro */
public Numeros(String num)
/** ajoute un numéro à la liste */
public void add(String num)
/** retourne le premier numéro de la liste (il existe forcément) */
public String numero()
/** retourne true si la liste contient le numéro donné */
public boolean has(String num)
/** retourne le nombre de numéros de la liste (>=1) */
public int count()
/** retourne la séquence des numéros séparés par des virgules dans une chaîne */
public String toString()
/** enlève le numéro donné de la liste. @require: count()>1 */
public void remove(String num)
```

Réaliser et tester (faire un programme de test) cette classe en choisissant successivement les classes «LinkedList» et «ArrayList» comme implémentation de « List ». Quel est le meilleur choix selon vous ?

2ème Partie : Modélisation de l'application Annuaire

La classe « Annuaire » représente la structure de donnée conservant les associations entre les personnes et leurs numéros de téléphones. On utilisera l'interface « Map » comme structure de donnée de l'annuaire.

3. Spécification de la classe Annuaire

```
/** notion d'annuaire : ensemble d'associations Personne-Numero */
public class Annuaire {
/** constructeur d'annuaire vide */
public Annuaire()
/**
* ajoute une nouvelle entrée dans l'annuaire. Si p n'existe pas: on crée une nouvelle
* association (p,n) ; sinon : on ajoute n aux numéros de p
* Correspondance interface: BOUTON Ajouter
*/
public void addEntry(Personne p, String n)
/**
* chargement de l'annuaire depuis un fichier texte (le contenu de l'annuaire est remplacé)
* chaque ligne du fichier est de la forme :
* Civilite Nom Prenom "Numero1" "Numero2" .... <CR> (Civilite = Mr ou Mme ou Melle)
* Correspondance interface: BOUTON Charger
*/
public void loadEntryFromFile(String file)
/**
* retourne le premier numéro de la personne, si la personne est absente retourne null
* Correspondance interface: BOUTON Numéro
*/
public String getNumber(Personne p)
/**
* retourne les numéros de la personne, si la personne est absente retourne null
* Correspondance interface: BOUTON Numéros
*/
public String getNumbers(Personne p)
/**
* retourne l'intégralité de l'annuaire dans un ordre quelconque : une personne par ligne
* suivie de ses numéros de téléphone
* Correspondance interface: BOUTON Print Répertoire
*/
public String toString()
/**
* retourne la première personne ayant le numero donné, null si aucune personne
* Correspondance interface: BOUTON Personne
*/
public Personne annuInverse(String num)
/**
* supprime la personne de l'annuaire, si elle est présente
* Correspondance interface: BOUTON Supprimer (si le champ "numero" est vide)
*/
public void remove(Personne p)
/**
* supprime le numero donné de la personne, s'il n'y a plus qu'un numéro dans la liste supprime la personne
* Correspondance interface: BOUTON Supprimer (si le champ "numero" est rempli)
*/
public void remove(Personne p,String num)
```

4. Scénario 1 : Réalisation d'une version de la classe Annuaire

Réaliser la classe Annuaire en choisissant la classe « HashMap » comme implémentation de «Map». Testez votre classe: importez un petit annuaire préalablement saisi dans un fichier de type texte, et notamment demandez l'affichage de l'annuaire.

5. Scénario 2 : Réalisation d'une version de la classe Annuaire

Modifier l'implémentation en choisissant la classe « TreeMap ». Quel problème cela pose-t-il ? Quelle doit être la propriété de la classe Personne ? Faites les modifications nécessaires, refaites le test précédent et constatez les modifications lors de l'affichage de l'annuaire.

6. Scénario 3 : Compléter la classe annuaire

On veut ajouter la méthode suivante à l'annuaire :

```
/**
* retourne la liste des personnes dont le nom commence par la lettre donnée (minuscule ou majuscule)
* (une personne par ligne, avec ses numéros)
*/
public String toString(char c)
```

Quelle interface choisir comme structure de donnée de la classe Annuaire pour simplifier l'écriture de cette méthode ? Quelle classe concrète pouvez-vous choisir comme implémentation ?

Réaliser cette évolution et tester cette nouvelle méthode.

7. Scénario 4 : Adaptation d'une solution

On veut pouvoir ajouter à l'annuaire des entrées fournies dans un fichier avec la logique suivante : si la personne existe déjà dans la table, on ajoute les numéros à ceux existant, sinon on ajoute l'entrée correspondante dans l'annuaire.

Réalisez la méthode spécifiée ci-dessous :

```
/**
* ajout des entrées contenues dans un fichier texte chaque ligne du fichier est de la forme :
* Civilite Nom Prenom "Numero1" "Numero2" ....
* Correspondance interface: BOUTON Importer
*/
public void addEntryFromFile(String file)
```

Proposez une solution pour éviter les doublons dans la liste des numéros.

8. Scénario 5 : Généralisation de la solution

On souhaite pouvoir s'affranchir de la nature effective des structures de données utilisées dans les classes Numeros et Annuaire. Pour cela il faut paramétrer cette information afin que le code de création que nous avons mis en place pour l'instant ne fasse plus directement référence au type effectif de l'objet à créer (Par exemple LinkedList dans la classe Numeros). On met à votre disposition, dans la classe Util, la méthode adéquate pour ce faire, sachant que l'on considère qu'il existe systématiquement un constructeur sans argument pour les classes choisies.

Ajouter dans les classes concernées l'attribut nécessaire pour fixer le nom de la classe à utiliser pour engendrer l'objet effectif et modifier le code en conséquence.

Afin d'éviter la compilation qui reste encore nécessaire, externalisez cette information en utilisant le package java.util.prefs.

Le mécanisme de préférence, proposé par le package java.util.prefs, permet de gérer des préférences au niveau système ou au niveau utilisateur. Pour initialiser vous pouvez utiliser les méthodes suivantes :

Method Summary	
static Preferences	**systemNodeForPackage**(Class<?> c) Returns the preference node from the system preference tree that is associated (by convention) with the specified class's package.
static Preferences	**systemRoot**() Returns the root preference node for the system.
static Preferences	**userNodeForPackage**(Class<?> c) Returns the preference node from the calling user's preference tree that is associated (by convention) with the specified class's package.
static Preferences	**userRoot**() Returns the root preference node for the calling user.

Si les méthodes sans arguments sont utilisées, il faut alors utiliser la méthode suivante pour obtenir les préférences :

abstract Preferences	**node**(String pathName) Returns the named preference node in the same tree as this node, creating it and any of its ancestors if they do not already exist.

Pour obtenir la valeur associée à une préférence il faut utiliser les méthodes suivantes où <x> représente un type primitif et <X> son wrapper :

abstract String	**get**(String key, String def) Returns the value associated with the specified key in this preference node.
abstract boolean	**get<X>**(String key, <x> def) Returns the boolean value represented by the string associated with the specified key in this preference node.

Le stockage de l'information est assuré par le système ; Sous le système Windows les préférences sont stockées dans la base de registres. Vous pouvez utiliser l'outil « preference tool » pour visualiser et/ou éditer l'état des préférences. Bien évidemment, sous windows, vous pouvez utiliser l'éditeur de la base de registres, la clé est soit HKEY_CURRENT_USER/Sotfware/javaSoft/Prefs soit HKEY-LOCAL-MACHINE/SOTWARE/javaSoft/Prefs en fonction que vous accédez aux préférences systèmes ou utilisateurs. Pour éviter d'avoir à accéder à la base de registres vous pouvez utiliser un fichier xml qui contient la définition des préférences que vous souhaitez avoir. Ceci permet aussi une modification plus simple. Vous pouvez utilisez la méthode suivante qui permet de charger une configuration contenue dans ce fichier xml.

static void	**importPreferences**(InputStream is) Imports all of the preferences represented by the XML document on the specified input stream.

Le contenu du fichier xml pourrait être de la forme :

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE preferences SYSTEM "http://java.sun.com/dtd/preferences.dtd">
<preferences EXTERNAL_XML_VERSION="1.0">
        <root type="user">
          <map/>
          <node name="jus">
            <map/>
            <node name="aoo">
              <map/>
              <node name="annuaire">
                <map>
                  <entry key="Personne/Type" value="java.util.ArrayList"/>
                </map>
              </node>
            </node>
          </node>
        </root>
</preferences>
```

3ème Partie : Extension de l'application

9. Adaptation de la classe Numeros

Les numéros constituent un ensemble car il n'est pas nécessaire d'avoir dans cette liste plusieurs fois le même numéro. Modifier la classe Numéros en utilisant ce qu'offre la librairie Java pour gérer des ensembles et en faisant en sorte de minimiser les modifications.

1. Indiquez les modifications à apporter à la classe Numeros.

On souhaite faire la même chose mais en utilisant le mécanisme d'héritage ?

2. Quel problème cela pose-t-il ? Comment palier à ce problème ? (question plus difficile)

10. Spécialisation de la classe Numero

Pour être proche de la réalité, nous proposons de considérer désormais qu'un numéro de téléphone n'est pas réduit à la chaîne de caractères (String) qui le compose. Développer un classe Numero qui représente ce concept en imaginant les principales fonctionnalités qui vous sembleraient utiles.

Commencez par une version élémentaire, par la suite vous pourrez prendre en compte la diversité incontournable des numéros de téléphones internationaux.

1. Si l'on substitue au type String le type Numero dans la classe Numeros, quel problème cela entraine-t-il ?

2. Que conclure sur le choix initial de représenter les numéros de téléphone par une chaîne de caractères ? Qu'aurait-il été plus pertinent de faire ?

3. Comment confiner la modification à la classe Numeros ? Réaliser cette solution.

11. Généralisation de la classe Annuaire

Pour permettre de mieux organiser son répertoire téléphonique, on souhaite introduire la possibilité d'avoir plusieurs annuaires qui permettrait d'avoir des catégories (par exemple : personnel, professionnel, …). Un sous-annuaire serait identifié par un nom unique et les opérations d'ajout et de recherche seraient possibles sur chacun des annuaires mais aussi sur l'ensemble des annuaires.

12. Amélioration de l'interfaçage avec la classe Util

La classe Util fournit une méthode d'acquisition d'annuaire. Celle-ci indique précisément qu'il doit exister une classe Personne ayant un constructeur d'une signature (profil) précise, et de même pour la classe Numeros.

Quel reproche peut-on faire à cette manière d'imposer les choses ?

Quelle solution naturelle devrait-on mettre en œuvre ? Pourquoi n'est-elle pas applicable ?

Proposez une solution. Pour la mettre en œuvre, il vous faut écrire une nouvelle version d'importFromFile en réalisant une sous-classe d'Util qui assure cette redéfinition.

4ème Partie : Conservation et restauration de l'annuaire

13. Production d'un fichier au format XML

Complétez la classe Annuaire dont le squelette vous est fourni et qui contient une méthode saveAnnuaire(Writer p) sauvegardant l'état courant, au format XML, de l'annuaire dans un fichier fourni en paramètre. Pour indication nous vous donnons la grammaire du langage de stockage (on note entre simple-quotes les terminaux, entre accolades ce qui est optionnel, la barre verticale indique le choix d'alternatives, Nom, Prenom et Chiffres sont des méta-terminaux donc compris entre double-quotes)

```
Annuaire ::= '<' 'annuaire' '>' Entrees '</' 'annuaire' '>'.
Entrees ::= Entree {Entrees}.
Entree ::=  '<' 'entree' {'civilite=' Civilité} 'nom=' Nom 'prenom=' Prenom '>'
                        Numeros
                    '</' 'entree' '>'.
Civilite ::= "'" ( 'Melle' | 'Mme' | 'Mr') "'".
Numeros ::= Numero {Numeros}.
Numero ::= '<' 'numero' 'tel=' Chiffres '/>'.
```

14. Réalisation d'un analyseur permettant de charger un fichier annuaire.

Utilisez la classe StreamTokenizer ou Scanner et les méthodes qui vous sont fournies (Terminal, IfTerminal et MetaTerminal) pour construire un analyseur de la grammaire définie au paragraphe 2.1 (méthode loadAnnuaire(Reader r)). On considère, pour simplifier le problème, que les textes fournis respectent la syntaxe.

Vérifier que vous pouvez charger le fichier XML produit précédemment.

15. Manipulation d'un fichier au format Objet (sérialisation)

On souhaite désormais stocker les annuaires dans le format propriétaire proposé au sein de Java. Pour cela on utilisera la sérialisation qui permet de stocker des objets dans un format externe non textuel pour autant qu'ils soient 'sérialisables'. Pour cela vous utiliserez les classes ObjectOutputStream et ObjectInputStream.

16. Sauvegarde au format "serializable"

Réaliser dans la classe Annuaire une méthode saveObject(ObjectOutput s) qui permet de sauvegarder l'état courant de l'annuaire dans le flot fourni en paramètre.

17. Acquisition au format "serializable"

Réaliser dans la classe Annuaire une méthode loadObject(ObjectInput s) qui permet de restaurer l'état courant de l'annuaire à partir du flot fourni en paramètre.