Associations et tables associatives

Exercices

Auteur : E.Thirion – 24/02/2019

Document extrait du site cours.thirion.free.fr

Les exercices suivants sont en majorité des projets à complèter. L'interface graphique de ces projets est déjà réalisée, ce qui vous permettra un gain de temps important. Ces projets sont disponibles par téléchargement. Pour savoir comment y accéder cliquez ici.

D'autre part, ce document fait partie d'un ensemble de cours du même auteur (programmation procédurale, programmation objet, programmation web, bases de données) auxquels vous pouvez accéder en cliquant ici.

Points commun aux exercices de ce document

Les exercices de ce document ont pour but de vous faire travailler simultanément les associations entre objets, la manipulation des tables associatives et l'utilisation de **this** en tant que référence à un objet. Nous reprenons ici les différentes associations vues en cours concernant les classes **Personne** et **Voiture**.

Une voiture est identifiée par son numéro d'immatriculation représenté par l'attribut **Immat** de type **String**:

```
public class Voiture {
  String Immat;
  ....
}
```

A une exception près (association partie d'échec) on supposera qu'une personne est identifiée par son prénom:

```
public class Personne {
  String prenom;
  ....
}
```

Les associations sont représentées par des attributs de ces deux classes dont la déclaration est imposée.

L'initialisation des données et des combo box suit le même principe que l'exemple de projet **Conduire1** détaillés dans le cours (voir associations [1-1] non réflexives) dont on pourra s'inspirer.

L'ensemble des voitures (respectivement des personnes) est contenu dans la table associative **LesVoitures** (respectivement **LesPersonnes)** indexée par leurs numéros d'immatriculation (respectivement par leurs prénoms).

Le code permettant de remplir ces deux tableaux avec des données est déjà écrit, de même que le code permettant de sélectionner une personne donnée ou une voiture donnée à partir de l'interface graphique.

Au départ aucune association n'existe entre les objets. Elles sont crées ou supprimées par l'utilisateur lorsqu'il clique sur les boutons prévus à cet effet. Votre rôle sera principalement de faire fonctionner ces boutons.

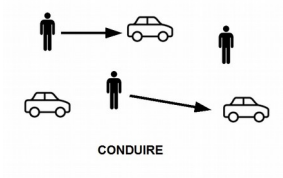
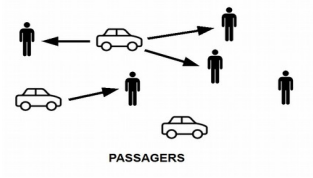
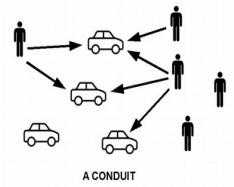
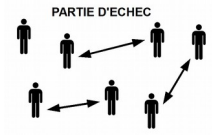
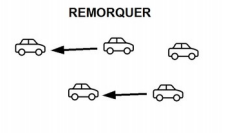
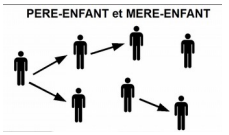
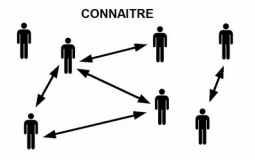
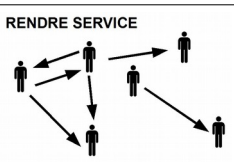
Attention: bien que ces exercices soient assez similaires, la manière de créer ou de supprimer une association est différente dans chacun d'eux car ils diffèrent tous par le type d'association traitée ou la manière de la représentée (voir le tableau récapitulatif à la page suivante).

Cas particulier de l'exercice 7 (Projet Parent)

Cet exercice est différent des autres à plusieurs points de vue:

- Il n'y a pas qu'une seule association à gérer, mais deux: l'association père-enfant et l'association mère-enfant.
- Il est plus long et plus complexe : il y a non seulement les boutons permettant de créer ou supprimer des associations, mais aussi des boutons permettant de déduire des associations non représentées en mémoire, comme les frères et soeurs, les oncles et tantes, les cousins et cousines etc ... Une occasion de pratiquer l'union de tables associatives avec la méthode **putAll**.

Résumé des différentes types d'association et exemples ou exercices correspondants

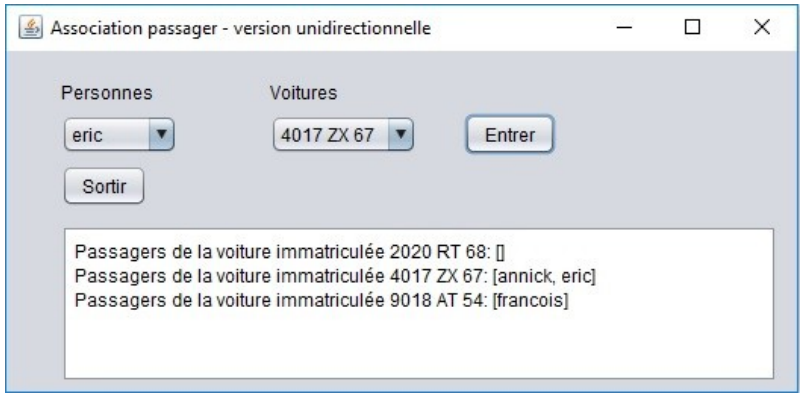
<table>
<tr><td rowspan="6">Non réflexives</td><td rowspan="2">[1-1]</td><td rowspan="2"></td><td rowspan="2"> 
</td><td>Version unidirectionnelle
Exemple détaillé dans le cours
Projet Conduire1</td></tr>
<tr><td>Version bidirectionnelle
Exemple détaillé dans le cours
Projet Conduire2</td></tr>
<tr><td rowspan="2">[1-N]</td><td rowspan="2"></td><td rowspan="2"> 
</td><td>Version unidirectionnelle
Exercice 1
Projet Passagers1</td></tr>
<tr><td>Version bidirectionnelle
Exercice 2
Projet Passagers2</td></tr>
<tr><td rowspan="2">[N-N]</td><td rowspan="2"></td><td rowspan="2"> 
</td><td>Version unidirectionnelle
Exercice 3
Projet A_Conduit1</td></tr>
<tr><td>Version bidirectionnelle
Exercice 4
Projet A_Conduit2</td></tr>
<tr><td rowspan="6">Réflexives</td><td rowspan="2">[1-1]</td><td>Symétrique</td><td> 
</td><td>Exercice 5
Projet Partie_Echec</td></tr>
<tr><td>Assymétrique</td><td> 
</td><td>Exercice 6
Projet Remorquer</td></tr>
<tr><td>[1-N]</td><td></td><td> 
</td><td>Exercice 7
Projet Parent</td></tr>
<tr><td rowspan="2">[N-N]</td><td>Symétrique</td><td> 
</td><td>Exercice 8
Projet Connaitre</td></tr>
<tr><td>Assymétrique</td><td> 
</td><td>Exercice 9
Projet Rendre_Service</td></tr>
</table>

Association non réflexives

Exercice 1 : L'association passager – version unidirectionnelle

Projet à ouvrir : **Exo_Java-ProgObjet2/Passagers1**

Fenêtre du projet



La zone de texte pour l'affichage des associations se nomme **ZT_Passager**.

Représentation des données

Dans la version unidirectionnelle, seule la classe **Voiture** contient un attribut pour représenter l'association. Il s'agit de l'attribut **LesPassagers** :

```
public class Voiture {
  String Immat;
  TreeMap <String,Personne> LesPassagers;
  ....
}
```

Cet attribut est prévu pour contenir l'ensemble des passagers de la voiture sous la forme d'une table associative indexée par leurs prénoms.

Question 1 : Affichage des associations

L'affichage des associations dans la zone de texte **ZT_Passager** est réalisée par la procédure **AfficherLesVoitures**. Il s'agit donc ici d'afficher les passagers de chaque voiture. On pourra utiliser ici la méthode **toString** de la classe **TreeSet** en l'appliquant aux clés de la table **LesPassagers**.

Question 2: Le bouton Entrer et méthodes associées

2-a) Voiture dans laquelle se trouve un passager

Comme la représentation est unidirectionnelle, on utilise une fonction pour retrouver la voiture dans laquelle se trouve une personne. Il s'agit de la fonction **VoitureDansLaquelleSeTrouve** qui prend comme paramètre un objet de la classe **Personne** et retourne un objet de la classe **Voiture**. Retournez **null** si la personne en

question ne se trouve dans aucune voiture.

2-b) La méthode Entrer

Cette méthode de la classe **Voiture** simule en quelque sorte l'action d'entrer dans une voiture. Elle prend en paramètre un objet de la classe **Personne**. En écrivant cette méthode, ne tenez pas compte de l'état des associations: on fera rentrer la personne dans la voiture quelque soient les conditions.

2-c) Le bouton Entrer

Au contraire, dans le bouton **Entrer** on tiendra compte de l'état des associations. Une personne ne peut entrer dans une voiture que si elle ne se trouve dans aucune voiture. Pour tester cette condition, on utilisera la fonction **VoitureDansLaquelleSeTrouve**. Pour faire entrer la personne dans la voiture utilisez la méthode **Entrer**. Si l'utilisateur tente de faire entrer une personne dans une voiture, alors que cette personne occupe déjà une voiture, on affichera un message d'erreur précisant l'immatriculation de la voiture occupée.

Question 3: Le méthode et le bouton Sortir

3-a) La méthode

Comme pour la méthode **Entrer**, complètez cette méthode de la classe **Voiture** afin de simuler l'action de sortir d'une voiture sans tenir compte de la possibilité physique de le faire.

3-b) Le bouton

Le bouton **Sortir** ne tient compte que de la personne sélectionnée. Son effet est d'éjecter en quelque sorte cette personne de la voiture dans laquelle elle se trouve. On utilisera pour cela la méthode **Sortir**. Si la personne ne se trouve dans aucune voiture, on affichera un message d'erreur.

Exercice 2 : L'association passager – version bidirectionnelle

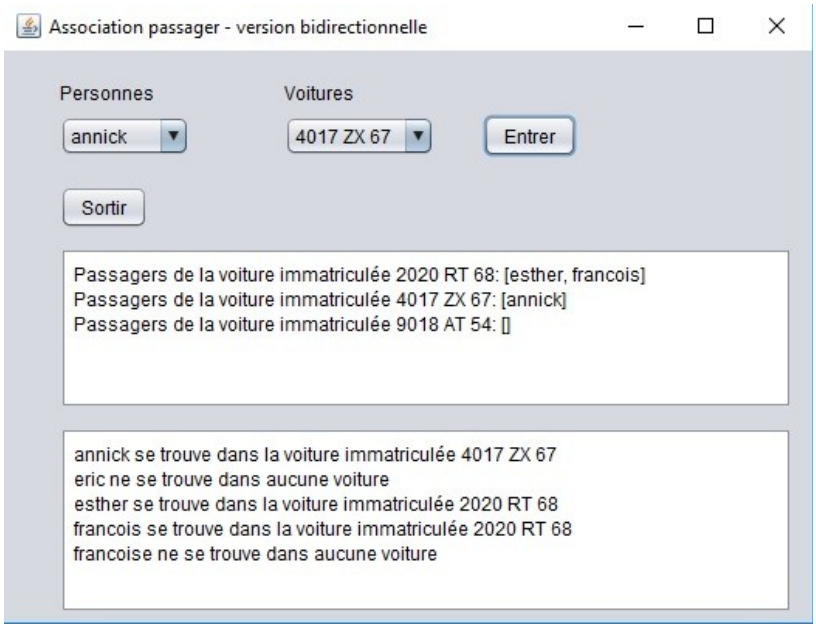
Représentation des données

Dans cette nouvelle version, la classe **Personne** contient un attribut-pointeur (**DansVoiture**) vers la voiture dans laquelle se trouve la personne. On supposera que la valeur de cet attribut est **null** lorsque la personne ne se trouve dans aucune voiture.

```
public class Personne {
   String prenom;
   Voiture DansVoiture;
   ...
```

Projet à ouvrir : **Exo_Java-ProgObjet2/Passagers2**

Fenêtre du projet



La zone de texte du haut se nomme **ZT_Passager** et celle du bas, **ZT_Voiture**.

Objectif: à vous de trouver les modifications à apporter pour faire fonctionner cette nouvelle version !

Exercice 3 : L'association A Conduit – Version unidirectionnelle

Représentation des données

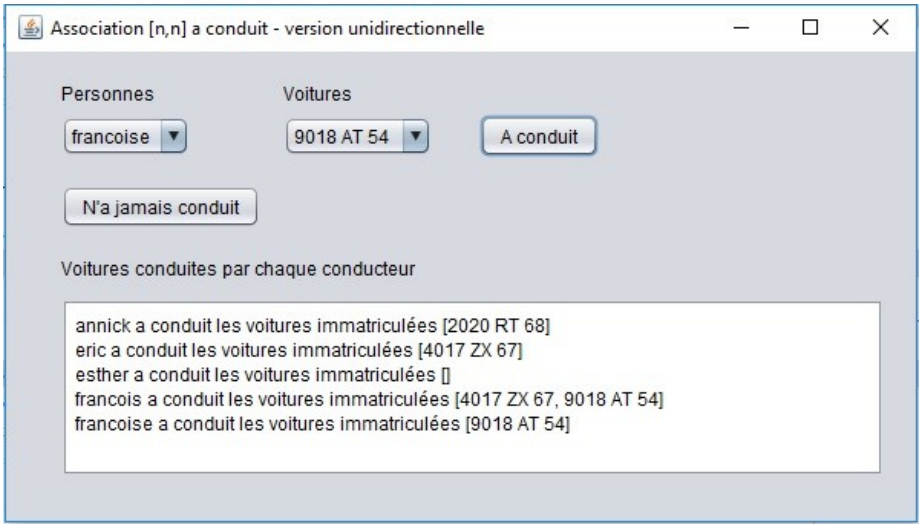
Dans la version unidirectionnelle, l'association **A Conduit** est représentée par l'attribut **A_Conduit** de la classe **Personne**:

```
public class Personne {
   String prenom;
   TreeMap <String,Voiture> A_Conduit;
   ...
```

Il contiendra toutes les voitures conduites par une personne sous la forme d'une table associative indexée par les numéros d'immatriculation des voitures.

Projet à ouvrir : **Exo_Java-ProgObjet2/A_Conduit1**

Fenêtre du projet



La zone de texte se nomme **ZT_A_Conduit**.

Question 1: Affichage des associations

Complètez la procédure **AfficherLesConducteurs** afin qu'elle affiche les voitures conduites par chaque personne dans la zone de texte.

Question 2: La méthode et le bouton A_Conduit

2-a) La méthode

Cette méthode se situe dans la classe **Personne**. Elle prend en paramètre un objet de la classe **Voiture**. Son effet est d'ajouter cette voiture dans l'ensemble des voitures conduites par la personne.

2-b) Le bouton

Il a pour effet d'ajouter la voiture sélectionnée à l'ensemble des voitures conduites par la personne sélectionnée. On ne vous demande pas de contrôler si la voiture est déjà dans l'ensemble.

Question 3: Le bouton N'a jamais conduit et la méthode associée

3-a) La méthode A_Jamais_Conduit

Cette méthode se situe dans la classe **Personne**. Elle prend en paramètre un objet de la classe **Voiture**. Son effet est de supprimer cette voiture de l'ensemble des voitures conduites par la personne.

3-b) Le bouton N'a jamais conduit

Il a pour effet de supprimer la voiture sélectionnée de l'ensemble des voitures conduites par la personne sélectionnée. On ne vous demande pas de contrôler si la voiture est bien dans l'ensemble.

Exercice 4 : L'association A Conduit – Version bidirectionnelle

Représentation des données

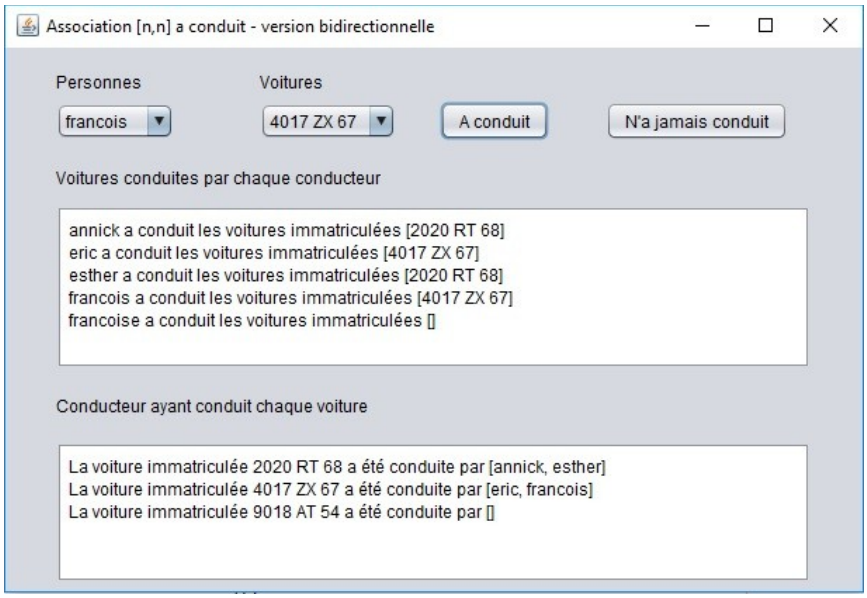
Dans la version bidirectionnelle, l'association **A Conduit** est représentée deux attributs: l'attribut **A_Conduit** de la classe **Personne,** ainsi que l'attribut **A_Ete_Conduite_Par** de la classe **Voiture**:

```
public class Voiture {
   String Immat;
   TreeMap <String,Personne> A_Ete_Conduite_Par;
   ...
```

Il contiendra toutes les personnes ayant conduit une voiture donnée sous la forme d'une table associative indexée par les prénoms des conducteurs.

Projet à ouvrir : **Exo_Java-ProgObjet2/A_Conduit2**

Fenêtre du projet



La zone de texte du bas a été rajoutée. Elle se nomme **ZT_A_Ete_Conduite**.

Objectif: à vous de trouver les modifications à apporter pour faire fonctionner cette nouvelle version !

Association réflexives

Exercice 5 : L'association Partie d'échec

Représentation des données

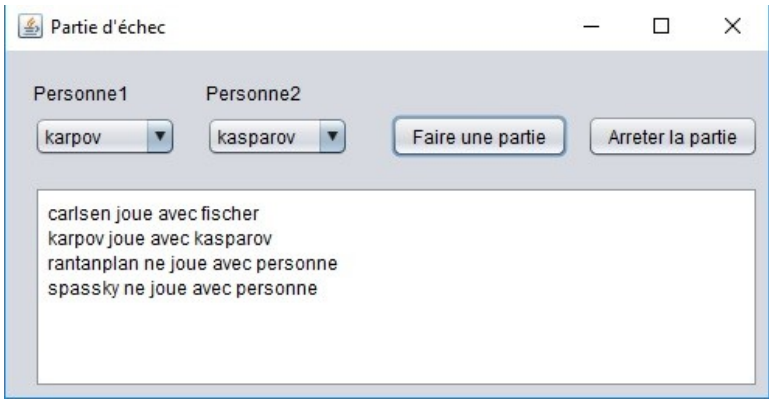
Nous somme ici en présence d'une association [1-1] réflexive et symétrique. Comme elle est réflexive, il n'y a qu'une seule classe:

```
public class Personne {
   String nom;
   Personne joue_avec;
   ...
```

Notez que dans cet exemple, une personne n'est plus identifiée par son prénom, mais par son nom. L'attribut **joue_avec** sert de pointeur vers l'adversaire.

Projet à ouvrir : Exo_Java-ProgObjet2/Partie_Echec

Fenêtre du projet



La zone de texte se nomme **ZT_Joue_Avec**.

Question 1: Affichage des associations

Il s'agit ici d'afficher qui joue avec qui et qui ne joue avec personne. Difficulté supplémentaire: essayez de ne pas afficher une association dans les deux sens. Par exemple, si on affiche "carlsen joue avec fischer", il est inutile de d'afficher également "fischer joue avec carlsen".

La méthode à complèter se nomme **AfficherLesParties**.

Question 2: La méthode et le bouton Faire_Une_Partie

2-a) La méthode

Cette méthode met en place une partie d'échec entre deux personnes, sans faire aucun contrôle.

2-b) Le bouton

Il s'agit ici d'appliquer la méthode **Faire_Une_Partie** à condition que:

1. les deux personnes soient distinctes. Si c'est le cas afficher un message d'erreur.
2. aucune des deux personnes ne soit déjà engagée dans une partie d'échec. Si c'est le cas préciser qui joue déjà avec qui.

Question 3: La méthode et le bouton Arreter_La_Partie

3-a) La méthode

Elle arrête la partie d'échec entre deux personnes, sans faire aucun contrôle.

3-b) Le bouton

Il s'agit ici d'appliquer la méthode **Arreter_La_Partie** à condition que:

1. les deux personnes soient distinctes.
2. Qu'il existe bien une partie d'échec entre les deux personnes concernées.

Si une de ces deux conditions n'est pas vérifiée, affichez un message d'erreur explicite.

Exercice 6 : L'association Remorquer

Représentation des données

Ici tout se passe dans la classe **Voiture**:

```
public class Voiture {
   String Immat;
   Voiture remorque;
   Voiture remorque_par;
   ...
```

Les deux attributs **remorque** et **remorque_par** représentent l'association de manière bidirectionnelle.

Projet à ouvrir : Exo_Java-ProgObjet2/Remorquer

Fenêtre du projet

La zone de texte se nomme **ZT_Remorquer**.

Question 1: Affichage des associations

Inspirez vous de la copie d'écran.

Question 2: La méthode et le bouton Remorquer

2-a) La méthode

Elle met en place une association de remorquage entre deux voitures, sans faire aucun contrôle.

2-b) Le bouton

On applique la méthode **Remorquer** à la voiture 1, afin qu'elle remorque la voiture 2, à condition que:

1. Les deux voitures ne soient pas identiques.
2. Aucune des deux voitures ne remorque une autre voiture
3. Aucune des deux voitures n'est remorquée par une autre voiture.

Afficher un message d'erreur explicite lorsqu'une de ces conditions n'est pas vérifiée.

Question 3: La méthode et le bouton ne_plus_remorquer

3-a) La méthode

Elle supprime l'association de remorquage entre deux voitures, sans faire aucun contrôle.

3-b) Le bouton

On applique la méthode **ne_plus_remorquer** à la voiture 1, afin qu'elle ne remorque plus la voiture 2, à condition que la voiture 1 remorque effectivement la voiture 2. Si ce n'est pas le cas, afficher un message d'erreur explicite.

Exercice 7 : Relations famillliales

Représentation des données

A la base des relations famillliales (oncle, tante, grand-père, cousins, etc ...) se trouvent la relation parent-enfant que l'on peut décomposer en deux associations **[1-N]** : l'association père-enfant et l'association mère-enfant. Nous avons choisi de représenter ces deux associations par les trois attributs **pere**, **mere** et **les_enfants** :

```
public class Personne {
   String prenom;
   boolean homme;
   Personne pere;
   Personne mere;
   TreeMap <String,Personne> les_enfants;
   ...
```

L'attribut booléen **homme** nous permet de différencier un homme d'une femme (**homme = true** signifie que la personne est du genre masculin).

Attenton: si **pere == null**, cela ne signifie pas que la personne n'a pas de père (ni que le père est nul !), mais que le père est inconnu (de même pour **mere == null**). Par contre **les_enfants == null** est interprèté comme une absence d'enfants.

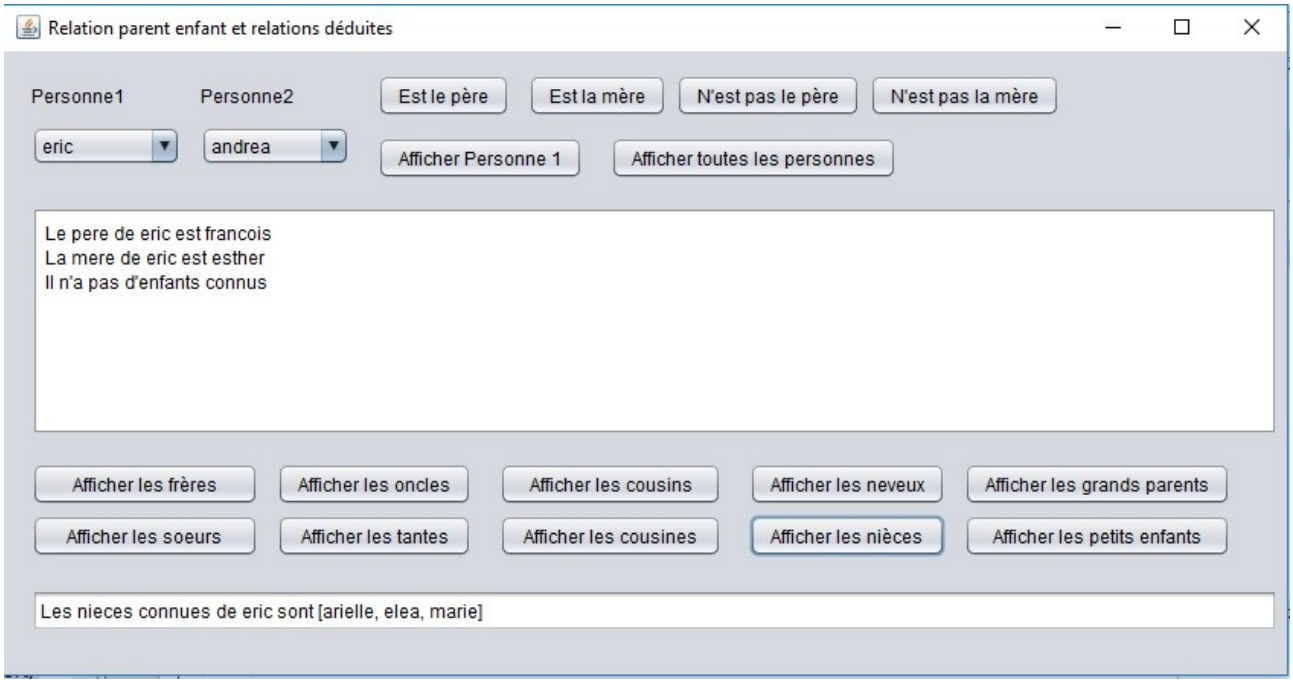
Pour pouvoir faire plus facilement les tests, je vous suggère d'entrer les prénoms des membres de votre

famille dans le code source. Pour cela, mettez tous les prénoms de votre famille dans le tableau **ToutLeMonde** et dans le tableau **LesHommes**, ne mettez que les prénoms des hommes. Si deux personnes portent le même prénom, ajoutez une lettre ou deux pour les distinguer (exemple: **eric-c** pour eric clapton et **eric-t** pour eric thirion).

Les personnes sont automatiquement créés à partir de ces deux tableaux par la procédure **CreerLesPersonnes** dont le code est donné. Elles sont stockées dans la table associative **LesPersonnes** indexée par les prénoms.

Projet à ouvrir : Exo_Java-ProgObjet2/Parent

Fenêtre du projet



La zone de texte se nomme **ZT_Relation_Parent_Enfant** et le champ de texte du bas, **CT_Relations_Deduites**.

Question 1: Affichage des associations parent-enfant d'une personne

Il s'agit ici de complèter la procédure **AfficherLesRelationsParentEnfantDe** qui prend en paramètre le prénom d'une personne, de manière à ce qu'elle affiche les parents et les enfants de cette personne dans la zone de texte **ZT_Relation_Parent_Enfant**. Plus précisément, l'affichage devra préciser si le père est inconnu ou sinon, donner son prénom. Idem pour la mère. Il devra également préciser si il (ou elle) n'a pas d'enfants connus ou sinon préciser les prénoms des enfants. Notez bien que le logiciel ne peut afficher que les enfants connus (c'est à dire déductibles des relations enfants enregistrées jusque là). Il ne pourra donc jamais « être certain » de connaitre la totalité des enfants d'une personne.

La procédure **AfficherLesRelationsParentEnfantDe** est utilisée dans les deux boutons intitulés « Afficher personne 1 » et « Afficher toutes les personnes ». Le code de ces deux boutons est donné. Ils vous permettrons de vérifier que votre code fonctionne correctement. Le premier bouton affiche uniquement les relations parent-enfant de la personne 1. Le deuxième affiche les relations parent-enfant de toutes les personnes connues.

Question 2: Gestion de l'association pere-enfant

2-a) La méthode est_le_pere

Cette méthode met en place une association pere-enfant entre deux personnes. Le père est la personne à

qui la méthode est appliquée et l'enfant, la personne passée en paramètre. Il s'agit ici uniquement de mettre à jour les valeurs des attributs **les_enfants** et **pere** sans contrôler la validité de l'association.

2-b) Le bouton « Est le père »

En cliquant sur ce bouton , l'utilisateur indique que la personne 1 sélectionnée est le père de la personne 2 sélectionnée. Il faudra donc ici mettre en place cette association en utilisant la méthode **est_le_pere** à condition que:

- les deux personnes soient distinctes
- la personne 1 soit bien de sexe masculin
- la personne 2 n'a pas déjà un père connu

Afficher un message d'erreur explicite, si une de ces conditons n'est pas vérifiée.

D'autre part, pour visualiser l'effet de ce bouton, affichez les relations parent-enfant des deux personnes concernée à l'aide de la procédure **AfficherLesRelationsParentEnfantDe**.

2-c) La méthode n_est_pas_le_pere

L'utilisateur peut se tromper en donnant une fausse relation père-enfant. La méthode **n_est_pas_le_pere** permet de la défaire. La personne qui n'est pas le père est celle à qui la méthode est appliquée et celle qui n'est pas l'enfant, la personne passée en paramètre. Comme pour la méthode **est_le_pere**, il s'agit ici uniquement de mettre à jour les attributs **les_enfants** et **pere**.

2-d) Le bouton « N'est pas le père »

En cliquant sur ce bouton , l'utilisateur indique que la personne 1 n'est pas le père de la personne 2. Il faudra donc ici supprimer cette association en utilisant la méthode **n_est_pas_le_pere** à condition que la personne 1 soit bien actuellement considérée comme étant le père de la personne 2.

Afficher un message d'erreur explicite si ce n'est pas le cas. Sinon afficher les nouvelles relations parent-enfant dans la zone de texte.

Question 3: Gestion de l'association mere-enfant

Mêmes sous-questions que pour l'association pere-enfant en remplacant la mère par le père.

Question 4: La procédure MaFamille

Afin de pouvoir tester le bon fonctionnement des boutons suivants, complètez la procédure **MaFamille** de telle sorte qu'elle génère les relations pere-enfant et mere-enfant de tous les membres votre famille (en y incluant vos grand parents, vos freres et soeurs, vos oncles, tantes, cousins et cousines, neveux, nièces). Utilisez pour cela les méthodes **est_le_pere** et **est_la_mere**. Cette procédure est appelée au démarrage du projet.

Question 5: Frères et soeurs

5-a) Détermination des enfants selon leur sexe

On vous demande ici de complèter la méthode **enfants_de_sexe** de la classe **Personne**. Elle prend en paramètre un booléen et retourne sous la forme d'une table associative indexée par les prénoms, tous les enfants de sexe masculin si le paramètre vaut **true** et tous les enfants de sexe féminin sinon. Cette méthode vous sera utile pour déterminer les frères et soeurs d'une personne.

5-b) La méthode freres

Cette méthode retourne les frères d'une personne sous la forme d'une table associative indexée par les prénoms. On suppose pour simplifier qu'il n'y a pas de demi-frere. Le frere d'une personne est donc un autre enfant de sexe masculin d'un de ses deux parents. Si aucun des deux parents n'est connu, on retournera la

valeur **null** (à distinguer le l'ensemble vide qui signifie que la personne n'a pas de frères !).

5-c) Le bouton « Afficher les frères »

Ce bouton permet d'afficher tous les frères de la personne 1, dans le champ de texte **CT_Relations_Deduites**. S'il n'est pas possible de déterminer les frères afficher un « ? », sinon afficher les prénoms des frères connus.

5-d) La méthode soeurs

Même principe que la méthode **freres**.

5-e) Le bouton « Afficher les soeurs »

Même principe que pour le bouton intitulé « Afficher les frères ».

Question 6: Oncles et tantes

6-a) La méthode oncles

Cette méthode retourne les oncles d'une personne sous la forme d'une table associative indexée par les prénoms. Comme notre logiciel ne connait pas le mariage, on supposera qu'il n'y a pas d'oncle par alliance, c'est à dire qu'un oncle est uniquement le frère d'un des deux parents.

Cette méthode va bien sur utiliser la méthode **freres**. Retourner **null** si aucun oncle ne peut être déduit des informations enregistrées. Indication: la méthode **putAll** pourra être utile pour cette question.

6-b) Le bouton « Afficher les oncles »

Ce bouton permet d'afficher tous les oncles de la personne 1, dans le champ de texte **CT_Relations_Deduites**. S'il n'est pas possible de déterminer les oncles afficher un « ? », sinon:

- afficher les prénoms des oncles connus, si la personne en a.
- préciser que la personne n'a aucun oncle connu, sinon.

6-c) La méthode tantes

Même principe que la méthode oncles.

6-d) Le bouton « Afficher les tantes »

Même principe que le bouton « Afficher les oncles ».

Question 7: Les cousins et les cousines

7-a) La méthode Oncles_Et_Tantes

Cette méthode retourne sous la forme d'une table associative indexée par les prénoms, l'ensemble des oncles et tantes d'une personne. Retourner **null** si aucun oncle et aucune tante ne peut être déterminée.

7-b) La méthode Cousins

Cette méthode retourne sous la forme d'une table associative indexée par les prénoms, l'ensemble des cousins d'une personne. Un cousin étant un enfant de sexe masculin d'un oncle ou d'une tante, on pourra utiliser ici la méthode précédente **Oncles_Et_Tantes**. Retourner **null** si aucun cousin ne peut être déterminé.

7-c) Le bouton « Afficher les cousins »

Ce bouton permet d'afficher tous les cousins de la personne 1, dans le champ de texte **CT_Relations_Deduites**. S'il n'est pas possible de déterminer les cousins afficher un « ? », sinon:

- afficher les prénoms des cousins connus, si la personne en a.
- dans le cas contraire, préciser que la personne n'a aucun cousin connu.

7-d) La méthode Cousines

Même principe que pour la méthode Cousins.

7-e): Le bouton « Afficher les cousines »

Même principe que pour le bouton « Afficher les cousins ».

Question 8: Les neveux et les nièces

8-a) La méthode freres_ou_soeurs

Cette méthode retourne sous la forme d'une table associative indexée par les prénoms, l'ensemble des frères ou des soeurs d'une personne. Retourner **null** si aucun frère ni aucune soeur ne peut être déterminée.

8-b) La méthode Neveux

Cette méthode retourne sous la forme d'une table associative indexée par les prénoms, l'ensemble des neveux d'une personne. Etant donné qu'un neveux est un enfant de sexe masculin d'un frere ou d'une soeur, on pourra utiliser ici la méthode de la question précédente. Retournez **null** si aucun cousin ne peut être déterminé.

8-c) Le bouton « Afficher les neveux »

Ce bouton permet d'afficher tous les neveux de la personne 1, dans le champ de texte **CT_Relations_Deduites**. S'il n'est pas possible de déterminer les neveux afficher un « ? », sinon:

- afficher les prénoms des neveux connus, si la personne en a.
- dans le cas contraire, préciser que la personne n'a aucun neveux connu.

8-d) La méthode Niece

Même principe que la méthode **Neveux**.

8-e) Le bouton « Afficher les nieces »

Même principe que pour le bouton « Afficher les neveux ».

Question 9: Grands parents et petits enfants

9-a) La méthode Grands_Parents

Cette méthode retourne sous la forme d'une table associative indexée par les prénoms, l'ensemble des grands parents connus d'une personne. Retourner **null** si aucun grand parent ne peut être déterminé.

9-b) Le bouton « Afficher les grands parents »

Ce bouton permet d'afficher tous les grands parents de la personne 1, dans le champ de texte **CT_Relations_Deduites**. S'il n'est pas possible de les déterminer afficher un « ? », sinon afficher les prénoms des grands parents connus.

9-c) La méthode Petits_Enfants

Retourne sous la forme d'une table associative indexée par les prénoms, l'ensemble des petits enfants

connus d'une personne.

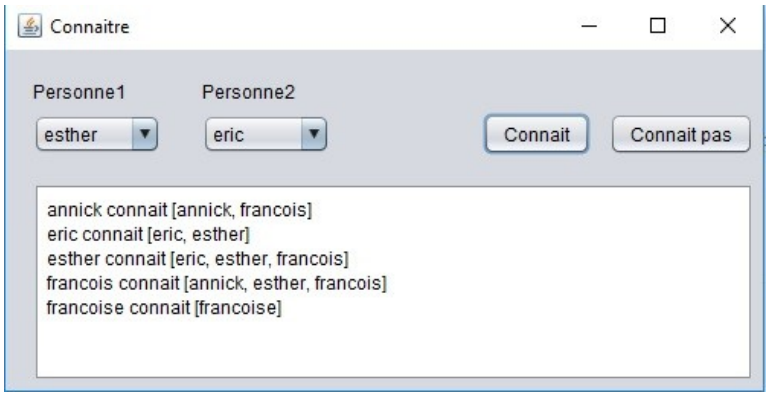
9-d) Le bouton « Afficher les petits enfants »

Ce bouton permet d'afficher tous les petits enfants de la personne 1, dans le champ de texte **CT_Relations_Deduites**. Si elle a des petits enfants, afficher leurs prénoms. Sinon, préciser qu'elle n'en a pas.

Exercice 8 : L'association Connaitre

Projet à ouvrir : Exo_Java-ProgObjet2/Connaitre

Fenêtre du projet



La zone de texte se nomme **ZT_Connaitre**.

Représentation des données

L'association connaître est une association [n-n] symétrique, par conséquent un seul attribut suffit à la représenter de manière bidirectionnelle. Ici, c'est l'attribut **connait** qui joue ce rôle:

```
public class Personne {
   String prenom;
   TreeMap <String,Personne> connait;
   Personne (String p) {
       prenom = p;
       connait = new TreeMap <String,Personne> ();
       connait.put(prenom, this);
   }
   ...
}
```

Notez qu'une personne se connait forcément elle même. Cette relation est donc mise en place dès le départ dans le constructeur de la classe.

Question 1 : Affichage des associations

Complètez la procédure **AfficherLesConnaissances** pour qu'elle affiche la situation de chaque personne: afficher qu'elle ne connait personne ou sinon les prénoms des personnes qu'elle connait.

Question 2: Le bouton Connait et la méthode associée

2-a) La méthode Connaître

Elle établit une connaissance avec la personne passée en paramètre.

2-b) Le bouton Connait

Ce bouton établit une relation de connaissance entre les deux personnes sélectionnée. On acceptera ici que l'utilisateur entre plusieurs fois la même information (la personne A connait la personne B) sans le lui signaler.

Question 3: Le bouton « Connait pas » et la méthode associée

3-a) La méthode Ne_Pas_Connaitre

Elle supprime la relation de connaissance avec la personne passée en paramètre.

3-b) Le bouton

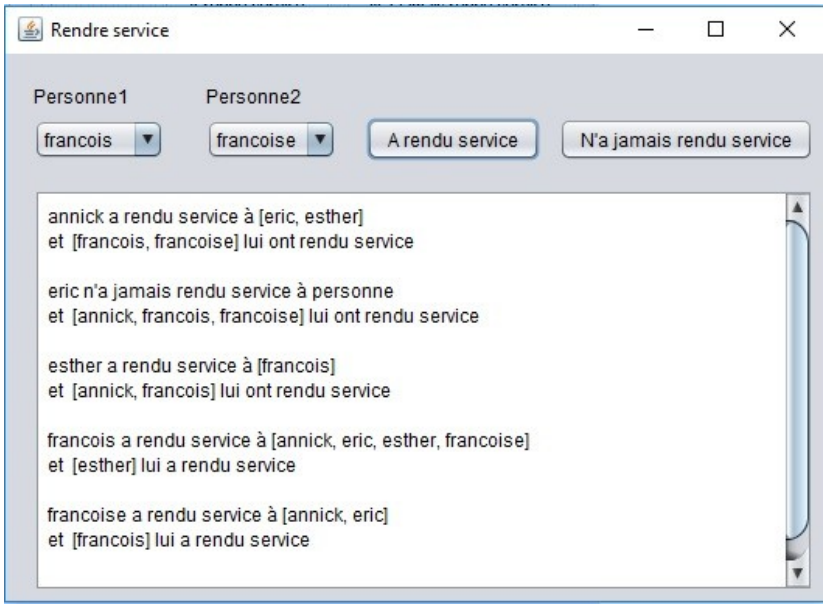
Il supprime la relation de connaissance entre les deux personnes sélectionnée. Un seul contrôle est demandé: vérifiez que l'utilisateur ne cherche pas à supprimer la connaissance de la personne avec elle même. Si c'est le cas, signalez lui qu'une personne se connait forcément elle même.

Vous n'avez pas besoin de vérifier que l'utilisateur cherche à supprimer une relation de connaissance inexistante.

Exercice 9 : L'association Rendre Service

Projet à ouvrir : **Exo_Java-ProgObjet2/Rendre_Service**

Fenêtre du projet



La zone de texte se nomme **ZT_Rendre_Service**.

Représentation des données

L'association **Rendre Service** est une association [n-n] assymétrique. Nous la représentons ici de manière bidirectionnelle avec deux attributs:

```
public class Personne {
   String prenom;
   TreeMap <String,Personne> a_rendu_service_a;
   TreeMap <String,Personne> ont_rendu_service;
   ....
}
```

Le premier, **a_rendu_service_a**, désigne l'ensemble des personnes à laquelle la personne représentée a rendu service. Le deuxième, **ont_rendu_service**, définit l'ensemble des personnes qui lui ont rendu service.

Question 1 : Affichage des associations

C'est la procédure **AfficherLesServices** qu'il s'agit de complèter ici. Pour chaque personne, afficher les personnes auxquelles elle a rendu service et inversement, les personnes qui lui ont rendu service. Plus précisément:

- Pour afficher afficher les personnes auxquelles elle a rendu service, distinguer deux cas:
  - il n'y en a aucune. Dans ce cas, afficher que cette personne n'a rendu service à personne.
  - Il y en a au moins une. Dans ce cas afficher les prénoms de ces personnes.
- Pour afficher les personnes qui lui ont rendu service, distinguer trois cas:
  - il n'y en a aucune. Dans ce cas, afficher que personne ne lui a rendu service.
  - Il y en a exactement une. Dans ce cas, afficher le prénom de cette personne suivi de « lui **a** rendu service ».
  - Il y en a plusieurs. Dans ce cas, afficher les prénoms de ces personnes suivis de « lui **ont** rendu service ».

Question 2: La méthode et le bouton « A rendu service »

2-a) La méthode

Appliquée à une personne **p**, elle ajoute la personne passée en paramètre à l'ensemble des personnes auxquelles **p** a rendu service.

2-b) Le bouton

Il ajoute la personne 2 à l'ensemble des personnes auxquelles la personne 1 a rendu service, à condition que ces deux personnes ne soient pas identiques (une personne ne peut pas se rendre service à elle même). Si c'est le cas afficher un message d'erreur.

Inutile de vérifier que l'utilisateur donne deux fois la même information.

Question 3: La méthode et le bouton « A jamais rendu service »

3-a) La méthode

Appliquée à une personne **p**, elle supprime la personne passée en paramètre de l'ensemble des personnes auxquelles **p** a rendu service.

3-b) Le bouton

Il supprime la personne 2 de l'ensemble des personnes auxquelles la personne 1 a rendu service. Aucune vérification n'est nécessaire.