TP JAVA – GESTION DE DROITS D'ACCES

L'intérêt principal de la gestion de droits d'accès est de mettre en œuvre en Java la programmation de relations entre classes et la notion de classe génériques.
Ces deux problématiques apparaissent dès la conception du diagramme de classes. Comme indiqué en **annexe 1**, nous avons choisi de mettre en place une classe d'association pour matérialiser la relation N-N qui apparaît dans le schéma. En ce qui concerne les classes conteneurs identifiées dans les données, nous avons considéré que l'on pouvait spécifier les classes conteneurs existantes pour répondre aux besoins spécifiques de notre application.

CLASSES GROUPEPERSONNE ET GROUPEPORTE

Les classes GroupePersonne et GroupePorte sont des classes qui contiennent respectivement des Personne et des Porte. A ce titre, il paraît intéressant de les rapprocher des classes conteneurs génériques offertes par le langage Java. On peut par exemple dire qu'un GroupePersonne est un Vector ayant un nom.

De plus, il apparaît clairement que pour ces 2 classes, il va nous falloir les mêmes opérations, à savoir ajouter une Personne ou une Porte, supprimer une Personne ou une Porte et enfin récupérer une Personne ou une Porte à partir d'une position donnée. Notre première idée a alors été de mettre en place une interface appelée Gerable qui aurait pu contenir les méthodes ajouterUnElement(), supprimerUnElement() et recupererUnElement().

En tenant compte de la première remarque (« un Groupe est un Vector ayant un nom ») et en regardant les méthodes de l'interface Gerable qu'auraient pu implémenter les classes GroupePersonne et GroupePorte, on se rend compte que le fait de faire hériter les deux classes précitées de Vector répond à toutes nos attentes. Le code source de GroupePersonne dans en **annexe 2** atteste de l'intérêt de cet héritage, très peu de lignes de code sont alors nécessaires. Un inconvénient cependant, il est possible dans l'état actuel de notre application d'insérer « n'importe quoi » dans les classes GroupePersonne et GroupePorte. Comme indiqué dans la méthode addElement() de la classe GroupePorte, il faudrait dans la redéfinition de cette méthode de Vector tester si l'Object passé en paramètre est une instance de Porte avant d'appeler la méthode de la super-classe Vector. On voit ici apparaît un des inconvénients de la généricité de Java …

CLASSE ACCES

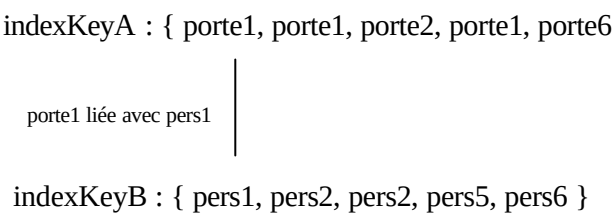
La relation entre les classes GroupePorte et GroupePersonne implique que l'on ait une structure qui stocke les paires de clés <instance de GroupePorte – instance de GroupePersonne>.

La première idée qui nous est venu à l'esprit a été de remettre en place le mécanisme vu l'an dernier en C++ (lors du projet Air France notamment). On insère dans GroupePorte un lien vers GroupePersonne, à savoir un Vector contenant les instances de GroupePersonne liées avec l'instance de GroupePorte. On effectue l'opération symétrique dans GroupePersonne.

Outre la lourdeur du maintien à jour de ces liens lors de l'utilisation, un inconvénient majeur réside dans le fait que les deux classes GroupePorte et GroupePersonne sont intimement liées. Si un jour l'on doit établir une association entre GroupePorte et une nouvelle classe GroupeSalle, la technique évoquée ci-avant impose de modifier le source de GroupePorte pour rajouter un lien vers GroupeSalle. On voit donc que cette mise en œuvre n'est pas des meilleures.

La solution que l'on a choisie est de mettre en place est une classe utilitaire Acces (présentée en **annexe 2**) qui permet de gérer l'association entre les deux classes GroupePorte et GroupePersonne. Plus généralement, la classe Acces peut servir à gérer des relations entre n'importe quelle classe. L'implémentation de la classe Acces peut porter à discussion : en effet, nous avons utilisé deux Vector, l'un contenant les clés provenant de GroupePorte et l'autre contenant les clés de GroupePersonne.

Par exemple, si indexKeyA sert à stocker les instances de GroupePorte et indexKeyB celles de GroupePersonne :



Ainsi, avec cet exemple, on constate que porte1 est liée avec pers1, pers2 et pers5. Réciproquement, on voit que pers2 est liée avec porte1 et porte2. En fait, la position de l'élément dans le Vector indexKeyA indique avec quel autre élément du Vector indexKeyB cet élément est lié.

On peut conclure en disant qu'une autre approche pour cette implémentation de la classe Acces est d'utiliser deux Hashtable de Vector comme proposé dans le partiel. Dans ce cas, on double le stockage des instances dans la classe - relation mais l'on dispose d'une plus grande souplesse d'utilisation …

ANNEXES

Le travail effectué dans ce TP est présenté dans les annexes qui suivent comme indiqué ci-dessous :

- **Annexe 1** : diagramme des classes UML.
- **Annexe 2** : code source des classes (par ordre alphabétique).
- **Annexe 3** : exemple d'utilisation de l'application (fonction main).
- **Annexe 4** : trace d'exécution.