Laboratoire 5

Les Collections, les interfaces Comparable et Comparator Java

Les Collections JavaFX

Objectifs d'apprentissage

- Utiliser structures de données de Java
- Utiliser la collection HashSet
- Trier une Collection
- Parcourir une collection à l'aide d'un itérateur
- Utiliser l'interface Comparable
- Utiliser l'interface Comparator
- Utiliser la collection JavaFX ObservableList
- Utiliser la collection JavaFX ObservaleMap

I- Les collections Java

1. Créez une classe *Document* qui contient :
- Les variables d'instances numéro (entier), titre et auteur
- Un constructeur qui initialise les variables d'instance à des valeurs reçues en arguments
- Une méthode *toString()* pour l'affichage d'un objet *Document*
- Les accesseurs nécessaires.

2. Créez une classe *PratiqueCollection* qui contient une méthode *main()* pour effectuer les opérations suivantes :

a) Déclarez une variable *listDoc* qui est un *ArrayList* d'objets de type *Document*

b) Créez les documents suivants :

```
Document d1 = new Document (1,"aabb", "auteur1");
Document d2 = new Document (2,"aaab", "auteur2");
Document d3 = new Document (3,"bbaa", "auteur3");
Document d4 = new Document (4,"abaa", "auteur4");
Document d5 = new Document (1,"aabb", "auteur1");
Document d6 = new Document (2,"aaaa", "auteur2");
Document d7 = new Document (2,"abaa", "auteur2");
```

c) Ajoutez tous ces documents à la liste *listDoc.* Ajoutez le document *d3* une deuxième fois.

d) Utilisez un itérateur, *Iterator<Document> it = listDoc.iterator()* pour afficher tous les documents présents dans la collection *listDoc.*
Vous pouvez aussi utiliser aussi une boucle *for* comme suit :
for (Document doc: listDoc){ System.out.println(doc); }

e) Utilisez la classe *HashSet* pour déclarer une variable *ensDoc* de type ensemble d'objets de type *Document* que vous initialisez avec la liste *listDoc*

f) Affichez les éléments

g) Avez-vous plus d'une fois le même document ? (même numéro, même titre et même auteur) Pourquoi ?

h) Dans la Classe *Document*, redéfinissez les méthodes *hahCode()* et *equals()* de la classe *Object* pour corriger le problème en g).
La méthode ***public boolean equals(Object unDocument)*** retourne *vrai* si les valeurs des attributs(numéro, titre et auteur) du document reçu en argument sont les mêmes que celles des attributs du document courant.
La méthode ***public int hashCode()*** peut être redéfinie en utilisant la somme des *hashCode()* des variables d'instance par exemple .
(new Integer(numero).hashCode() + titre.hashCode()+ auteur.hashCode()).

i) Réaffichez l'ensemble *ensDoc* et vérifier que vous n'avez plus de doublons

j) Pour trier la liste de documents *listDoc*, tapez l'instruction Collections.sort(listDoc)

k) L'instruction en j) ne compile pas. Bien lire le message d'erreur pour comprendre le problème.

l) Pour corriger le problème en k), implantez l'interface *Comparable* dans la classe *Document* de façon à trier la liste de documents par *titre*.

m) Réaffichez la liste des documents pour vérifier votre tri.

n) Pour pouvoir trier selon d'autres champs, il faut créer une classe qui implémente l'interface *Comparator.* Créez une classe *TriParAuteur* pour Trier selon l'auteur. Utilisez seulement les méthodes *compare()* et *compareTo()* dans votre classe.

o) Utilisez l'instruction *Collections.sort(listDoc,**new** TriParAuteur())* pour effectuer le tri.

p) Affichez la liste de documents pour observer le résultat

q) Créez une classe *triParAuteurEtParTitre*, pour trier selon l'auteur et le titre. Les titres doivent être triés en ordre décroissant. Utilisez seulement les méthodes *compare()* et *compareTo()* dans votre classe.

r) Réaffichez la liste pour observer le résultat du tri.

II- Les collections JavaFX

1. La collection JavaFX ObservableList

Créez une classe PratiqueJavaFXObservableList avec une méthode main() où vous devez :

- Déclarez un objet *ObservableList* à partir d'un objet *ArrayList* comme suit :
  ArrayList<String> listDoc = new ArrayList<String>();
  ObservableList<String> listeObservable = FXCollections.observableList(listDoc);

- Enregistrez l'écouteur suivant auprès de la liste :

```
listeObservable.addListener(new ListChangeListener< String>() {

    @Override
    public void onChanged(ListChangeListener.Change<? extends String> c) {
        // TODO Auto-generated method stub
        System.out.println("modification détectée: " + c);
        while(c.next()) {
            System.out.println("Ajout?" + c.wasAdded() );
            System.out.println("suppression?" + c.wasRemoved());

        }
    } });
```

- Ajoutez "chaine1" à la liste observable
- Ajoutez "chaine2" à la liste listDoc
- Ajoutez "chaine3" à la liste observable
- Retirez "chaine1"de la liste observable.
- Affichez la liste listDoc
- Affichez la liste observable
- Exécutez. Quels messages sont affichés par l'écouteur et quelles sont les opérations qui provoquent cet affichage?

2. La collection JavaFX ObservableMap

Créez une classe PratiqueJavaFXObservableMap avec une méthode main() où vous devez :

- Déclarer une un objet *ObservableMap* à partir d'un objet *Map* comme suit :
  HashMap<Integer, Document> map = new HashMap<Integer, Document>();
  ObservableMap<Integer, Document> mapObervable = FXCollections.observableMap(map);

- Enregistrez l'écouteur suivant auprès de la liste :

```
mapObervable.addListener( new MapChangeListener<Integer, Document>() {

    @Override
    public void onChanged(MapChangeListener.Change<? extends Integer, ? extends Document> c)
        // TODO Auto-generated method stub
        System.out.println("modification détectée: " + c);

        System.out.println("Ajout?" + c.wasAdded() );
        System.out.println("suppression?" + c.wasRemoved());
    }
});
```

- Créez 3 objets Document d1, d2 et d3
- Ajoutez les documents d1, d2 et d3 à la map observable
- Retirez le document d2 de la map observable.
- Utilisez la méthode *values()* de la classe *HashMap* pour afficher la liste *mapObservalbe*
- Exécutez votre programme.