

## TP 8 : TP\_Comparator

### Objectifs :

Assimiler les notions liées aux collections de type *List* et la notion de relation d'ordre mise en œuvre avec une implémentation de *Comparator*.

### Déroulement :

Vous allez dans ce TP travailler sur une collection de type *List<Etudiant>* en tant que variable d'instance d'une classe *Societe* permettant de modéliser le concept d'une liste d'étudiants qu'une société recrute ou congédie.

Cette collection sera ensuite exploitée pour produire l'affichage des étudiants recrutés selon un ordre croissant ou décroissant . Cette relation d'ordre sera celle que vous coderez en tant que scénario de tri.

---

### Etapes chronologiques à réaliser

- Créez une classe *Etudiant* avec les variables d'instance suivantes :

```
String prenom  
String nom  
int age
```

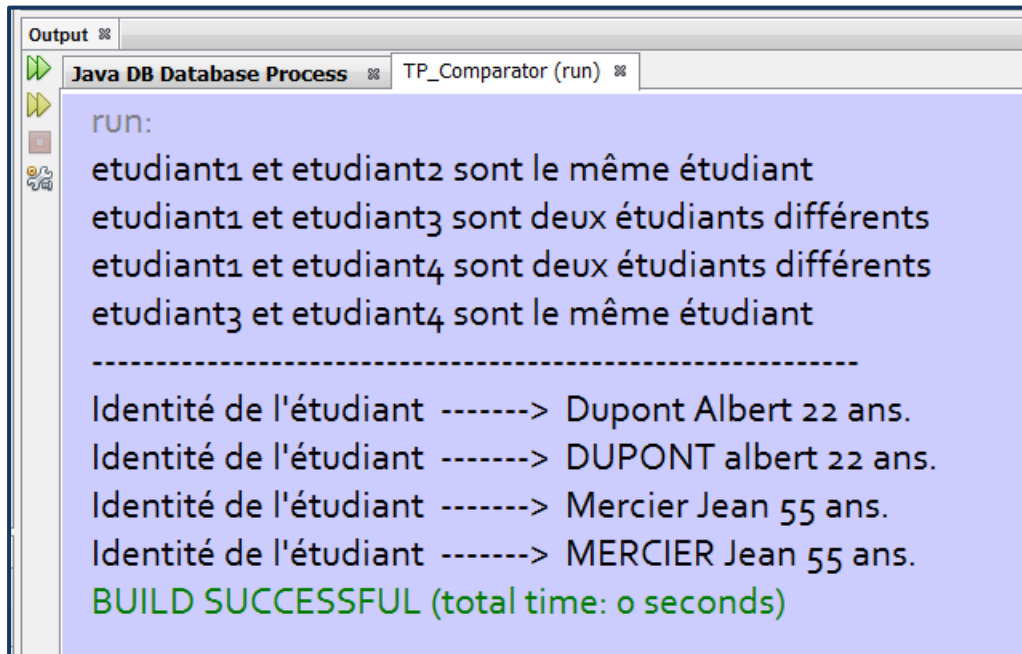
- Ajoutez les *getters/setters*.
- A l'aide de l'assistant , redéfinissez les méthodes *equals()* et *hashCode()* de façon à pouvoir affirmer que deux étudiants sont « égaux » lorsque toutes les valeurs de leurs propriétés ( nom, prenom et âge ) sont identiques. Vous reprendrez la méthode redéfinie *equals()* générée par l'assistant de façon à ignorer la casse pour les nom et prénom . Exemple :

```
Etudiant etudiant1 = new Etudiant("Dupont", "Albert", 22);  
Etudiant etudiant2 = new Etudiant("DUPONT", "albert", 22);
```

sont « le même étudiant » selon nos critères (Mêmes nom, prénom et âge).

- Redéfinissez également la méthode *toString()* pour produire les affichages de la figure 12 suivante.
- Instanciez quelques étudiants et comparez-les entre eux grâce à la méthode *equals()* que vous avez redéfinie :

```
Etudiant etudiant1 = new Etudiant("Dupont", "Albert", 22);
Etudiant etudiant2 = new Etudiant("DUPONT", "albert", 22);
Etudiant etudiant3 = new Etudiant("Mercier", "Jean", 55);
Etudiant etudiant4 = new Etudiant("MERCIER", "Jean", 55);
```



```
Output
Java DB Database Process TP_Comparator (run)
run:
etudiant1 et etudiant2 sont le même étudiant
etudiant1 et etudiant3 sont deux étudiants différents
etudiant1 et etudiant4 sont deux étudiants différents
etudiant3 et etudiant4 sont le même étudiant
-----
Identité de l'étudiant -----> Dupont Albert 22 ans.
Identité de l'étudiant -----> DUPONT albert 22 ans.
Identité de l'étudiant -----> Mercier Jean 55 ans.
Identité de l'étudiant -----> MERCIER Jean 55 ans.
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 12 : Affichage de quelques comparaisons et de quelques étudiants

- Créez ensuite une classe *Societe* qui va collectionner des étudiants qu'elle va recruter ou congédier s'ils ne conviennent pas :
- Déclarez une variable d'instance de type `ArrayList<Etudiant> listePersonnel`.
- Ajouter les méthodes suivantes, toujours dans la classe *Societe* :
  - *void* *recruter* (*Etudiant*)
  - *void* *congedier* (*Etudiant*)
- Au sein de ces deux méthodes, ajoutez dans un cas et supprimez dans l'autre l'étudiant en question dans la liste du personnel.
- Ajoutez un message dans *congedier(...)* de type : Bernard Morin quitte l'entreprise ...
- Ajoutez à *Societe* la méthode *afficher()* qui, moyennant un *Iterator*, affiche chacun des étudiants présents dans la collection *listePersonnel*.
- Créez des étudiants , une société et recrutez, congédiez des étudiants.
- Affichez la liste du personnel :

```

public static void main( String[] args ) {

    Societe entreprise = new Societe();
    Etudiant pers = new Etudiant ( "Durand", "Michel", 55);

    entreprise.recruiter(new Etudiant( "Mercier", "Jean", 50 ));
    entreprise.recruiter( new Etudiant( "Morin", "Nathalie", 35 ));
    entreprise.recruiter( new Etudiant( "Martin", "Louis", 35 ));
    entreprise.recruiter( new Etudiant( "Dupont", "Josette", 25 ));
    entreprise.recruiter( new Etudiant( "Charpentier", "Pierre", 25 ));
    entreprise.recruiter( pers );
    entreprise.afficher();
    entreprise.congедier( pers );
}

```

L'exécution du jeu d'essais proposé ci-dessus produit l'affichage suivant :

```

Output - TP_Comparator (run)
run:
-----
Identité de l'étudiant -----> Mercier Jean 50 ans.
Identité de l'étudiant -----> Morin Nathalie 35 ans.
Identité de l'étudiant -----> Martin Louis 35 ans.
Identité de l'étudiant -----> Dupont Josette 25 ans.
Identité de l'étudiant -----> Charpentier Pierre 25 ans.
Identité de l'étudiant -----> Durand Michel 55 ans.
-----
Michel Durand quitte l'entreprise ...
BUILD SUCCESSFUL (total time: 0 seconds)

```

- On souhaite maintenant pouvoir établir **un critère de tri** des étudiants lors de l'affichage : on va faire appel pour cela à la disponibilité suivante dans la classe utilitaire *Collections* :
- Consultez la **JavaDoc** pour revoir ces principes :

```
static <T> void sort ( List<T> list, Comparator< ? super T> c)
```

Sorts the specified list according to the order induced by the specified comparator.

- La méthode *sort* trie la liste fournie dans le 1<sup>er</sup> argument selon **la relation d'ordre** fournie par le *comparator* du 2<sup>ème</sup> argument .

- Vous allez donc devoir d'abord créer une classe qui implémente l'interface *Comparator* et, dans sa méthode *compare*, établir le critère d'ordre.
- Appelez cette classe *CompareEtudiant*.
- Ce critère d'ordre est le suivant : on trie selon le nom croissant, à nom identique, on trie selon le prénom, à prénom identique, on trie selon l'âge.
- Votre classe implémentant *Comparator* aura l'en-tête suivant, en utilisant la généricité, à vous de compléter :

```
public class CompareEtudiant implements Comparator<Etudiant> {

    @Override
    public int compare(Etudiant p1, Etudiant p2) {
        if ...
    }
}
```

- Il vous suffira ensuite de créer une instance de *CompareEtudiant* qui sera en conséquence le **comparateur** à fournir à la méthode *sort* :

```
Collections.sort ( liste_personnel, comparateur ) ;
```

- Ainsi, *sort* va utiliser votre scénario fourni via *comparateur* pour réorganiser la liste.
- On souhaite aussi pouvoir trier selon l'ordre décroissant, **inverse** du scénario ci-dessus.
- Vous allez pour cela utiliser la méthode *reverseOrder* de *Collections* qui propose de fournir un **comparateur** dotée d'une logique de tri inverse du *Comparator* que vous lui fournissez.
- Créez donc une nouvelle méthode *afficher ( boolean ordre )* qui, selon la valeur du paramètre *ordre* va afficher la liste dans un ordre croissant avec une instance de la classe *CompareEtudiant* créée ci-dessus ou en décroissant une instance « inverse » d'un point de vue tri de la classe *CompareEtudiant*.
- La méthode *afficher ( boolean ordre )* va donc commencer à examiner la valeur du booléen pour produire le bon ordre de tri , **croissant** ou **décroissant**.
- Il s'agit donc d'instancier un *CompareEtudiant* ou un autre selon une logique inversée . Voici comment instancier le bon comparateur **avant de trier** :

```
Comparator<Etudiant> comparateur = ordre ? new CompareEtudiant() :
    Collections.reverseOrder( new CompareEtudiant() );

Collections.sort ( liste_personnel, comparateur ) ;
```

- Générez un jeu d'essai avec des noms identiques , prénoms identiques , âges différents et identiques , bref toutes les situations envisageables et testez votre tri dans les deux sens .

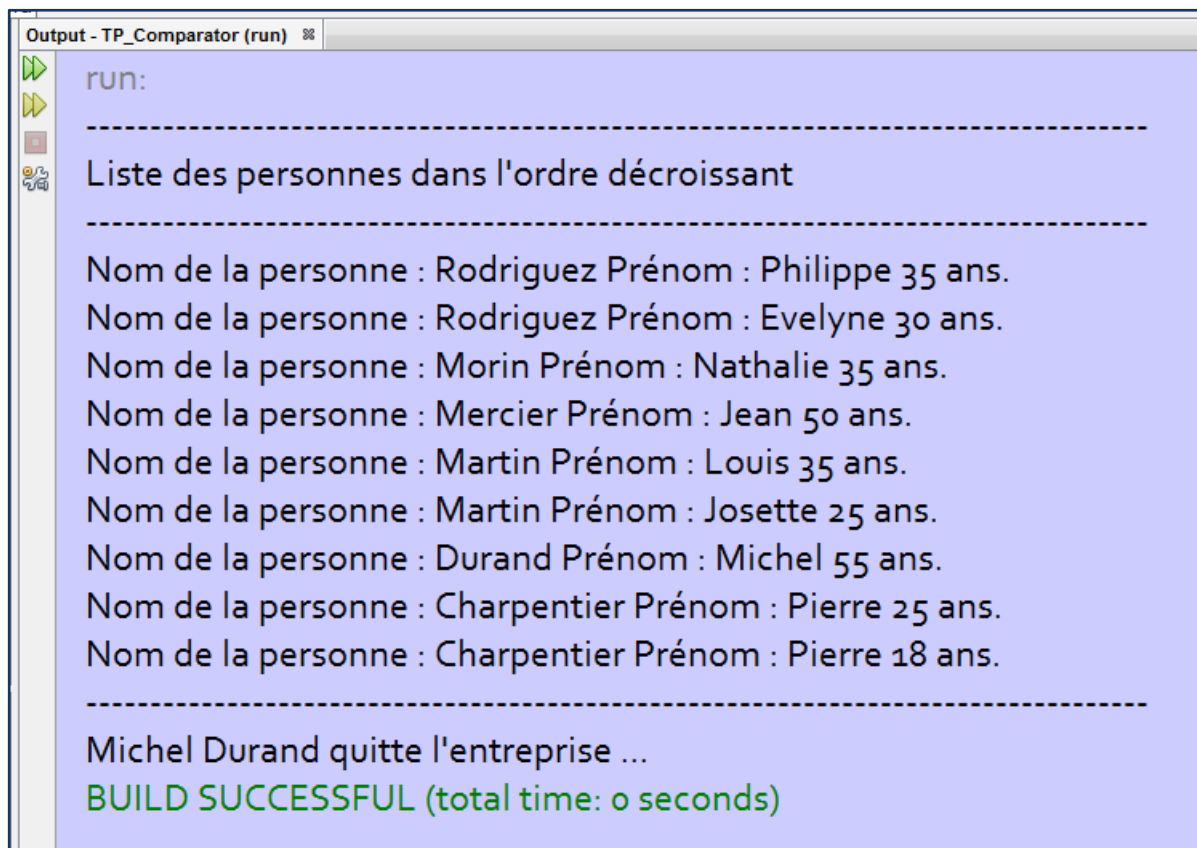
Exemple pour un tri croissant : *afficher ( true )* :

```
public static void main( String[] args ) {  
  
    Societe entreprise = new Societe();  
  
    Etudiant pers = new Etudiant ( "Durand", "Michel", 55 );  
    entreprise.recruiter( new Etudiant( "Mercier", "Jean", 50 ) );  
    entreprise.recruiter( new Etudiant( "Morin", "Nathalie", 35 ) );  
    entreprise.recruiter( new Etudiant( "Martin", "Louis", 35 ) );  
    entreprise.recruiter( new Etudiant( "Rodriguez", "Philippe", 35 ) );  
    entreprise.recruiter( new Etudiant( "Rodriguez", "Evelyne", 30 ) );  
    entreprise.recruiter( new Etudiant( "Martin", "Josette", 25 ) );  
    entreprise.recruiter( new Etudiant( "Charpentier", "Pierre", 25 ) );  
    entreprise.recruiter( new Etudiant( "Charpentier", "Pierre", 18 ) );  
  
    entreprise.recruiter( pers );  
    entreprise.afficher( true ); // Tri croissant  
}
```

A nom égal, on trie selon le prénom. A prénom égal , on trie selon l'âge.

```
Output - TP_Comparator (run) ✖  
run:  
-----  
Liste des personnes dans l'ordre croissant  
-----  
Nom de la personne : Charpentier Prénom : Pierre 18 ans.  
Nom de la personne : Charpentier Prénom : Pierre 25 ans.  
Nom de la personne : Durand Prénom : Michel 55 ans.  
Nom de la personne : Martin Prénom : Josette 25 ans.  
Nom de la personne : Martin Prénom : Louis 35 ans.  
Nom de la personne : Mercier Prénom : Jean 50 ans.  
Nom de la personne : Morin Prénom : Nathalie 35 ans.  
Nom de la personne : Rodriguez Prénom : Evelynne 30 ans.  
Nom de la personne : Rodriguez Prénom : Philippe 35 ans.  
-----  
Michel Durand quitte l'entreprise ...  
BUILD SUCCESSFUL (total time: 0 seconds)
```

## Exemple pour un tri décroissant : *afficher ( false )* ;



```
run:
-----
Liste des personnes dans l'ordre décroissant
-----
Nom de la personne : Rodriguez Prénom : Philippe 35 ans.
Nom de la personne : Rodriguez Prénom : Evelyne 30 ans.
Nom de la personne : Morin Prénom : Nathalie 35 ans.
Nom de la personne : Mercier Prénom : Jean 50 ans.
Nom de la personne : Martin Prénom : Louis 35 ans.
Nom de la personne : Martin Prénom : Josette 25 ans.
Nom de la personne : Durand Prénom : Michel 55 ans.
Nom de la personne : Charpentier Prénom : Pierre 25 ans.
Nom de la personne : Charpentier Prénom : Pierre 18 ans.
-----
Michel Durand quitte l'entreprise ...
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Constat : le tri avec le comparateur **direct** ou **inversé** fonctionne bien .
- Vous allez maintenant enrichir la **modélisation de la liste** : chaque étudiant qui est recruté devient un salarié. Symétriquement , un salarié qui est congédié redevient un étudiant.
- Chaque salarié possède un numéro d'identifiant qui garantit son unicité.
- Un nouveau critère de tri, résultant de la création de cette notion de salarié est demandé : Dans l'**ordre croissant** : on affiche d'abord les salariés et ce, selon leur numéro d'identification croissant, puis on affiche ensuite les étudiants selon les mêmes critères que dans l'activité précédente que vous venez de mettre en œuvre.



On peut remarquer que la liste contient désormais deux catégories d'instances : des *Etudiant* et des *Salarie*.

Commencez donc par créer la classe *Salarie* qui hérite de la classe *Etudiant* . Elle possède à ce titre *nom*, *prenom* et *âge*. Ajoutez la variable d'instance *identifiant* de type *int* .

- Le constructeur de *Salarie* requiert donc 4 arguments : trois qui sont transmis au constructeur de *Etudiant* et le 4<sup>ième</sup> qui sert à valoriser la variable d'instance *identifiant*.
- Concernant la classe d'implémentation de *Comparator* nécessaire au nouveau tri : créez une nouvelle classe à partir de *CompareEtudiant* (elle en hérite). Appelez-la : *CompareEtudiantSalarie*.
- En se souvenant que l'on va devoir comparer des étudiants et des salariés présents dans la liste, examinons le début de sa définition. Etant en présence potentielle d'instances d'*Etudiant* et de *Salarie*, il faut distinguer tous les cas possibles. Utilisez pour cela l'opérateur *instanceof* qui renvoie le type d'une instance.

```
public class CompareEtudiantSalarie extends CompareEtudiant {

    @Override
    public int compare( Etudiant etudiant1 , Etudiant etudiant2 ) {

        if ( etudiant1 instanceof Salarie && !( etudiant2 instanceof Salarie ) ) {
            return -1;
        } else

        if ( !( etudiant1 instanceof Salarie ) && ( etudiant2 instanceof Salarie ) )
            return +1;
        else

        if ( etudiant1 instanceof Salarie && etudiant2 instanceof Salarie ) {
            ....
        }
        ....
    }
}
```

- Dans la classe *Societe* , déclarez une variable de classe représentant le dernier numéro attribué au dernier salarié . Pensez à l'incrémenter à chaque nouvelle instanciation dans la méthode *recruter( Etudiant etudiant )*.
- Il est important de bien comprendre la signature des méthodes *recruter(...)* et *congedier(...)*

```
public Salarie recruter( Etudiant etudiant ) { // Renvoie un Salarie
    ...
}

public void congedier( Salarie employe ) {
    ...
}
```

- Instanciez plusieurs *Etudiants*. Recrutez-les. Congédiez quelques-uns d'entre eux .
- ... Et afficher dans l'ordre croissant puis décroissant , les personnes ( Etudiants et Salairés ) de la liste .

```
public static void main(String[] args) {  
  
    Societe societe = new Societe();  
  
    Salarie durand = societe.recruiter(new Etudiant("Durand", "Michel", 55));  
    Salarie leclerc = societe.recruiter(new Etudiant("Leclerc", "Claude", 50));  
    Salarie mercier = societe.recruiter(new Etudiant("Mercier", "Paul", 38));  
    Salarie dubois = societe.recruiter(new Etudiant("Dubois", "Paul", 38));  
    Salarie martin = societe.recruiter(new Etudiant("Martin", "Catherine", 21));  
    Salarie charpentier = societe.recruiter(new Etudiant("Charpentier", "Pierre", 25));  
    Salarie vannier = societe.recruiter(new Etudiant("Vannier", "Guillaume", 38));  
    Salarie rodriguez = societe.recruiter(new Etudiant("Rodriguez", "Isabelle", 25));  
  
    societe.congedier(vannier);  
    societe.congedier(martin);  
  
    Salarie duval = societe.recruiter(new Etudiant("Duval", "Isabelle", 30));  
    societe.congedier(duval);  
  
    societe.afficher(true);  
    societe.afficher(false);  
    System.out.println("-----");  
}
```

Le code ci-dessus produit l'affichage suivant :



```
Output - TP_Comparator (run)
run:
-----
Liste des salariés dans l'ordre croissant f( identifiant )
-----
Salarié N° 1 : Durand Michel 55 ans.
Salarié N° 2 : Leclerc Claude 50 ans.
Salarié N° 3 : Mercier Paul 38 ans.
Salarié N° 4 : Dubois Paul 38 ans.
Salarié N° 6 : Charpentier Pierre 25 ans.
Salarié N° 8 : Rodriguez Isabelle 25 ans.
Etudiant : Duval Isabelle 30 ans.
Etudiant : Martin Catherine 21 ans.
Etudiant : Vannier Guillaume 38 ans.
-----
Liste des salariés dans l'ordre décroissant f( identifiant )
-----
Etudiant : Vannier Guillaume 38 ans.
Etudiant : Martin Catherine 21 ans.
Etudiant : Duval Isabelle 30 ans.
Salarié N° 8 : Rodriguez Isabelle 25 ans.
Salarié N° 6 : Charpentier Pierre 25 ans.
Salarié N° 4 : Dubois Paul 38 ans.
Salarié N° 3 : Mercier Paul 38 ans.
Salarié N° 2 : Leclerc Claude 50 ans.
Salarié N° 1 : Durand Michel 55 ans.
-----
BUILD SUCCESSFUL (total time: 0 seconds)
```



Félicitations : vous avez réussi cet exercice dédié à la mise en œuvre de l'interface *Comparator*.