I Utilisation des fonctions de tri

Exemple

Objet	Un exemple d'utilisation des fonctions de tri.
Référence	Formation esup-commons V1.3.9
Version	1.0

A. Les fonctions de tri

JSF propose des fonctions de tri associées à divers TagLib.

L'inconvénient de ces fonctions, c'est qu'une fois utilisées dans un paginateur, le tri ne s'effectue que sur la page en cours.

Il faut donc prévoir de trier la liste qui alimente le paginateur avant l'affichage.

Pour se faire, nous pouvons utiliser les fonctions de tri des Collections Java.

Il existe 2 fonctions de tri dans Collections : *sort(List)* et *sort(List,Comparator)*.

1. *sort(List)*

Cette fonction trie la liste passée en paramètre en fonction de leur "ordonnancement naturel".
Tous les éléments de la liste doivent implémenter l'interface **Comparable**, et donc redéfinir la fonction *compareTo(Object)*, qui renvoie :

- 0 si les deux objets sont égaux,
- -1 si l'objet appelant est "inférieur"
- +1 si l'objet appelant est "supérieur"

Par exemple :

```
public class Personne implements Comparable
{
  public String nom,prenom;
  Personne(String nom, String prenom)
  {
    this.nom=nom;
    this.prenom=prenom;
  }
  public int compareTo(Object o) // on redéfinit compareTo(Object)
  {
    Personne p=(Personne)o;
    if(nom.equals(p.nom))
    {
      return prenom.compareTo(p.prenom);
    }
   return nom.compareTo(p.nom);
  }
}
```

Supposons ensuite une liste l de Personne, pour la trier, il suffira d'appeler *Collections.sort(l)*.

2. *sort(List,Comparator)*

Cette fonction trie la liste passée en paramètre en fonction du *Comparator* passé en paramètre. Supposons la classe Personne précédemment définie(sans implémenter Comparable).
Il faut créer une classe **implémentant l'interface Comparator**, et donc **redéfinissant** les fonctions *compare(Object,Object)* et *equals(Object)*, comme cet exemple :

```
import java.util.Comparator;
public class MonComparator implements Comparator
{
  MonComparator()
  {
  }
  public int compare(Object arg0, Object arg1)
  {
    Personne p1 = (Personne) arg0;
    Personne p2 = (Personne) arg1;

    int result = p1.name.compareTo(p2.name);

    if(result==0)
      result = p1.prenom.compareTo(p2.prenom);

        return result;
  }

}
```

l

l suffit alors, pour trier ma liste l, d'appeler *Collections.sort(l,new MonComparator()).*

B. Tri dans un paginateur

Dans la classe Paginator

```
	/**
	 * Le type de tri
	 */
	public String tri = "croissant";


	/**
	 * Le champ sur lequel on applique un tri
	 */ mis à jour depuis la JSF
	public String champTri = "nom";

Les accesseurs (getter et setter)
	/**
	 * @return the champTri (Le champ sur lequel on applique un tri)
	 */
	public String getChampTri(){
		return champTri;
	}

	/**
	 * @param String the champTri to set
	 */
	public void setChampTri(final String champTri){
		this.champTri = champTri;
	}

Les 2 méthodes (croissant et décroissant)

	public String triCroissant() {
		if (this.champTri.equals("nom"))
		{
			Collections.sort(listeAllProjet, new NameComparator());
		}
```

```
            else if (this.champTri.equals("date"))
            {
                  Collections.sort(listeAllProjet, new DateComparator());
            }
            tri = "croissant";
            return "";
      }


      public String triDecroissant() {
            if (this.champTri.equals("nom"))
            {
                  Collections.sort(listeAllProjet, Collections.reverseOrder(new
NameComparator()));
            }
            else if (this.champTri.equals("date"))
            {
                  Collections.sort(listeAllProjet, Collections.reverseOrder(new
DateComparator()));
            }
            tri = "decroissant";
            return "";
      }
```

C. Les comparateurs

1. NomComparator

```
public class NameComparator implements Comparator {

	@Override
	public int compare(Object o1, Object o2) {
		// TODO Auto-generated method stub
		String chaine1 = ((Projet)
o1).getInfoGen().getNom().toLowerCase().replace("é", "e").replace("è", "e");
		String chaine2 = ((Projet)
o2).getInfoGen().getNom().toLowerCase().replace("é", "e").replace("è", "e");
		if (chaine1.compareTo(chaine2)>0)  return 1;
		else if(chaine1.equals(chaine2)) return 0;
		else return -1;
	}

}
```

2. DateComparator

```
public class DateComparator implements Comparator {

	@Override
	public int compare(Object o1, Object o2) {
		// TODO Auto-generated method stub
	    Date date1 = ((Projet) o1).getInfoGen().getDateMiseOeuvre();
	    Date date2 = ((Projet) o2).getInfoGen().getDateMiseOeuvre();
	    if (date1.compareTo(date2)>0)  return 1;
	    else if(date1.equals(date2)) return 0;
	    else return -1;
	}

}
```

D. Dans la JSF

```
<f:facet name="header">
<h:panelGroup>
<e:text value="#{msgs['MESSAGE.FINALITE']}" />
      <t:commandButton image="#{tagsConfigurator.mediaPath}/picto-ecil-01.png"
      immediate="true" forceId="tri" alt="Down"
      action="#{GestionProjetController.paginator.triCroissant}">
      <t:updateActionListener value="nom"
      property="#{GestionProjetController.paginator.champTri}" />
      </t:commandButton>

      <t:commandButton image="#{tagsConfigurator.mediaPath}/picto-ecil-02.png"
      immediate="true" forceId="tridec" alt="Up"
      action="#{GestionProjetController.paginator.triDecroissant}">
      <t:updateActionListener value="nom"
      property="#{GestionProjetController.paginator.champTri}" />
      </t:commandButton>
</h:panelGroup>
</f:facet>
```