Tutoriel sur les collections Génériques en Java

ArrayList

- Collection ordonnée selon l'ordre d'insertion
- Tableau dynamique extensible à volonté.
- Accepte les doublons
- Accepte l'élément null
- Les éléments d'un ArrayList sont accessibles par leur indice. L'index commence à partir de 0.

```
public class ArrayListExemple {

	public static void main(String[] args) {

		ArrayList<Integer> l1=new ArrayList<Integer>();

		//1. Ajout d'éléments
		l1.add(10);
		l1.add(100);
		l1.add(40);
		// l1.add("jack");//erreur
		// l1.add(5.2);//erreur

		//2. imprimer une référence sur la liste
		System.out.println("Liste 1 ="+l1);

		//3. autorise les doublons
		l1.add(100);
		l1.add(100);
		System.out.println("Liste 1 ="+l1);

		//4. Récupération d'un élément par son index
		System.out.println(l1.get(2));
		//5. Modifier un élément de la liste
		l1.set(3, 50);
		l1.set(4, 70);

		//5. Parcours avec une boucle for indexée
		System.out.println("Parcours avec une boucle for indexée");
		for (int i=0;i<l1.size();i++){
```

```
                System.out.println(l1.get(i));

            }

            //6. trie de la liste
            Collections.sort(l1);
            System.out.println("Affichage après trie");
            for (int i=0;i<l1.size();i++){
                  System.out.println(l1.get(i));

            }

//7. Suppression d'un élément de la liste (faire attention au
débordement)
            l1.remove(l1.size()-1);
            //Suppression en début de liste LENTE
            l1.remove(0);

            //Parcours avec for each
                        System.out.println("Parcours avec une boucle
foreach");
                        for (Integer valeur : l1){
                              System.out.println(valeur);
                        }


            //interface List
            List <String> l2 =new ArrayList<>();
            l2.add("chien");
            l2.add("mouton");
            l2.add("rat");
            l2.add("mouton");
            //Parcours avec iterator
            System.out.println("Parcours avec iterator");
            Iterator<String> it=l2.iterator();
            while(it.hasNext()){
                  System.out.println(it.next());
            }
            // la méthode contains
            if (l2.contains("rat"))
                  System.out.println("le rat est dans la liste");
            else System.out.println("cet élément n'est pas dans la
liste");

            //Supression avec iterator
            System.out.println("Suppression avec iterator");
            Iterator<String> it2=l2.iterator();
            while(it2.hasNext()){
                  System.out.println(it2.next());
                  if(it2.next().equals("mouton"))
                        it2.remove();

            }

      }
```

LinkedList

- Est une liste ordonnée
- Liste doublement chainée
- Peut être parcouru par un ListIterator qui permet de parcourir la liste dans les deux sens.

```
public class LinkedListExemple {
	public static void main(String[] args) {
		LinkedList<Integer> ll1=new LinkedList<Integer>();

		//ajout d'éléments à la liste
		ll1.add(50);
		ll1.add(30);
		ll1.add(80);
		//Parcours avec une boucle for indexée
		System.out.println("Parcours avec une boucle for indexée");
		for (int i=0;i<ll1.size();i++){
			System.out.println(ll1.get(i));

		}

		//Ajout en tête et en fin de liste (présence de doublon)
		ll1.addFirst(100);
		ll1.addLast(50);

		//Parcours avec for each
		System.out.println("Parcours avec une boucle foreach");
		for (Integer valeur : ll1){
			System.out.println(valeur);
		}

		//Suppression du dernier élément
		ll1.removeLast();
		System.out.println("après supression du dernier élément");

		//parcours avec un ListIterator
		System.out.println("Parcours avec un ListIterator");
		ListIterator<Integer> i=ll1.listIterator();
		while(i.hasNext()){
			System.out.println(i.next());
		}


		 System.out.println("Parcours inverse avec un
ListIterator");
		 while(i.hasPrevious()){
				System.out.println(i.previous());
			}

		}

	}
```

Set : HashSet , LinkedHashSet et TreeSet

- Les Set sont des collections qui n'acceptent pas les doublons
- HashSet ne propose aucune garantie sur l'ordre de parcours lors de l'itération sur les éléments qu'elle contient (non ordonnée)
- LinkeSet est un Set qui maintient les clés dans l'ordre d'insertion
- TreeSet garantit que les éléments sont rangés dans leur ordre naturel **(interface Comparable)** ou l'ordre d'un Comparator.

```
public class SetExemple {

	public static void main(String[] args) {
		//une collection de type Set n'accepte pas les doublons

		//HashSet ne retient aucun ordre pour ses éléments
		Set<String> s1=new HashSet<String>();

		s1.add("chien");
		s1.add("chat");
		s1.add("vache");
		s1.add("chèvre");
		System.out.println(s1);
		//ajout d'un doublon ne fonctionne pas
		s1.add("chat");
		System.out.println("Affichage des éléments d'un HashSet");
		System.out.println(s1);

		//LinkedHashSet retient l'ordre d'insertion des éléments
		Set<String> s2=new LinkedHashSet<String>();
		s2.add("bonbon");
		s2.add("chocolat");
		s2.add("glace");
		s2.add("gateau");
	System.out.println("Affichage des éléments d'un LinkedHashSet");
		System.out.println(s2);


		//TreeSet ordonne les éléments selon leur ordre naturel
		Set<String> s3=new TreeSet<String>();
		if(s3.isEmpty())
			System.out.println("le Set 3 est vide au début");

		s3.add("veste");
		s3.add("manteau");
		s3.add("chemise");
		s3.add("jupe");

		if(s3.isEmpty())
			System.out.println("le Set 3 est vide après ajout");
```

```
            System.out.println("Affichage des éléments d'un TreeSet");
            System.out.println(s3);


            System.out.println("Affichage des éléments avec for each");
            ////////////itérations//////////
            for(String element:s3){
                  System.out.println(element);

            }

            ///Recherche d'un élément dans un set
            if(s1.contains("azerty"))
                  System.out.println("Le Set 1 contient l'élément
azerty");

            if(s3.contains("jupe"))
                  System.out.println("Le Set 3 contient l'élément
jupe");



            // s5 contient des éléments en commun avec S1
            Set<String> s5=new LinkedHashSet<String>();
            s5.add("ours");
            s5.add("chat");
            s5.add("vache");
            s5.add("mouton");

            /////// intersection  //////////
            System.out.println("Intersecrion des Set s1 et s5");
            Set<String> intersection=new LinkedHashSet<String>(s1);
            System.out.println(intersection);
            intersection.retainAll(s5);
            System.out.println(intersection);

      ///////Différence entre S1 et S5
            System.out.println("difference des Set s1 et s5");
            Set<String> diff=new LinkedHashSet<String>(s1);
            System.out.println(diff);
            diff.removeAll(s5);
            System.out.println(diff);


      }

}
```

Map : HashMap

- Les Map sont des collections sous la forme d'association clé-valeur
- La clé doit être unique.
- la même valeur peut être associée à plusieurs clés différentes.
- HasMap n'est pas ordonné
- LinkedHashMap conserve l'ordre d'insertion des éléments
- TreeMap stocke les éléments selon l'ordre naturel ou avec un comparator

```
public class HashMapExemple {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		HashMap<Integer,String> hm=new HashMap<Integer,String>();

		hm.put(5,"cinq");
		hm.put(2,"deux");
		hm.put(8,"huit");
		hm.put(3,"trois");
		hm.put(10,"dix");
		System.out.println(hm);


		//Ajout d'une clé existante
		hm.put(2,"zouuz");
		System.out.println(hm);


		String text=hm.get(2);
		System.out.println(text);

		//récupération de la valeur d'une clé inexistante
		System.out.println(hm.get(9));


		System.out.println("Parcours des clés");
		Set s= hm.keySet();
	       Iterator i1=s.iterator();
	       while(i1.hasNext())
	       System.out.print (i1.next()+" ");

	       System.out.println("\n Parcours des valeurs");
			Collection c= hm.values();
		       Iterator i2=c.iterator();
		       while(i2.hasNext())
		       System.out.print (i2.next()+" ");

		 System.out.println("\n Parcours d'un ensemble de paires
clé/valeur");
```

```
            //parcours des paires clé/valeur avec une boucle foreach
            System.out.println("parcours des paires clé/valeur avec une
boucle foreach");
            for (Entry<Integer,String> entree : hm.entrySet())
                  System.out.println(entree.getKey()+" :
"+entree.getValue());

            //parcours des paires clé/valeur avec un itérateur
            System.out.println("parcours des paires clé/valeur avec un
itérateur");
            Set<Entry<Integer, String>> s1=hm.entrySet();
            Iterator<Entry<Integer, String>> i=s1.iterator();
            while(i.hasNext()){
                  System.out.println(i.next());
            }

      }

}
```

Comparator

```
class StringLongueurComparator implements Comparator<String>{

      @Override
      public int compare(String s1, String s2) {
            int l1=s1.length();
            int l2=s2.length();
            if(l1==l2)
            return 0;
            else if(l1>l2)
                  return 1;
            else return -1;
      }
}
class AlphabetiqueComparator implements Comparator<String>{

      @Override
      public int compare(String s1, String s2) {
            //compareTo est une méthode de l'interface Comparable
            //Comparable correspond à l'implantation d'un ordre naturel
d'une classe
            return s1.compareTo(s2);
      }

}

class inverserAlphabetiqueComparator implements Comparator<String>{
```

```
	@Override
	public int compare(String s1, String s2) {
		//compareTo est une méthode de l'interface Comparable
		//Comparable correspond à l'implantation d'un ordre naturel
d'une classe
		return -s1.compareTo(s2);
	}

}


public class ComparatorExemple {

	public static void main(String[] args) {

		//////////////////////Sorting
String/////////////////////////
		List<String> animaux=new ArrayList<String>();
		animaux.add("brebis");
		animaux.add("zebre");
		animaux.add("rat");
		animaux.add("oiseau");
		animaux.add("mouton");
		animaux.add("lion");
		//sort permet de trier les éléments selon leur ordre naturel
		//Collections.sort(animaux);

		//Ordonne la liste selon le comparateur passé en paramètre à
la méthode sort

		//Collections.sort(animaux, new StringLongueurComparator());

		//Collections.sort(animaux,new AlphabetiqueComparator());

		//Collections.sort(animaux,new
inverserAlphabetiqueComparator());*/

	/*for(String animal : animaux){
			System.out.println(animal);
		}*/

//////////////////////Sorting numbers/////////////////////////
		List<Integer> nombres=new ArrayList<Integer>();
		nombres.add(15);
		nombres.add(9);
		nombres.add(72);
		nombres.add(18);
		nombres.add(23);
		nombres.add(45);
		//ordonne la collection selon l'ordre naturel
		Collections.sort(nombres);


		//ordonne la collection selon l'ordr inverse
		System.out.println("trier les nombres selon l'ordre
inverse");
		Collections.sort(nombres, new Comparator<Integer>() {
```

```
                @Override
                public int compare(Integer n1, Integer n2) {
                        return -n1.compareTo(n2);
                }

        });


        for(Integer n : nombres){
                System.out.print(n+" ");
        }
        System.out.println();

//////////////////////Sorting Object//////////////////////
        /*List<Personne> lesGens=new ArrayList<Personne>();
        Personne p1=new Personne(1,"zaineb");
        Personne p2=new Personne(2,"aymen");
        Personne p3=new Personne(3,"mouna");
        Personne p4=new Personne(4,"salah");
        lesGens.add(p3);
        lesGens.add(p4);
        lesGens.add(p1);
        lesGens.add(p2);

        //Collections.sort(lesGens);

        /* System.out.println("comparaison selon l'id");
          Collections.sort(lesGens, new Comparator<Personne>() {

                public int compare(Personne per1, Personne per2) {
                        if (per1.getId()>per2.getId())
                                return 1;
                        else if (per1.getId()<per2.getId())
                                return -1;
                        return 0;
                }
        });*/

      /*    System.out.println("comparaison selon l'ordre naturel des
noms");
        Collections.sort(lesGens, new Comparator<Personne>() {

                @Override
                public int compare(Personne o1, Personne o2) {
                        return o1.getNom().compareTo(o2.getNom());
                }

        });

        for (Personne p:lesGens){
                System.out.println(p);
        }

*/    }}
```