Coompléments sur les collections Java et couts des principales méthodes
Attention : toutes les méthodes ne sont pas listées

			Choses à respecter pour que ça marche			*Méthodes déclarées dans l'interface Collection<E>*							*Méthodes déclarées dans l'interface List<E>*						*Méthodes déclarées dans l'interface Queue<E>*			*Méthodes déclarées dans l'interface Deque<E>*		*Méthodes déclarées dans l'interface SortedSet<E>*		
Principales classes des collections Java	**Principales interface(s) réalisées par la classe**	**Quelle est la structure de donnée sous jacente ?**	**Egalité sementique entre deuxéléments E** ***boolean equals(Object o)* dans la classe E** Héritée de Object Peut être redéfinie dans E	**Capacité à calculer une valeur de hachage pour E** ***int hashCode()*** Héritée de Object Peut être redéfinie dans E *Doit être cohérente avec equals()*	**Relation d'ordre entre objets de type E** ***Soit** "ordre naturel"* E implements Comparable<E> (méthode int compareTo(E) dans la classe E) *Doit être cohérente avec equals()* ***Soit** petite classe implantant l'interface Comparator<E> (=>capable de compérer deux objets E)* eg : class MonCpmparator implements Comparator<E> (méthode int compare(E e1, E e2)	**add(E)**	**boolean contains(Object o)**	**boolean remove(Object)**	**int size()** **boolean isEmpty()**	**Récupérer un Iterator : Iterator<E> Iterator()**	**Remarque parcours par Itérateur** *l'itération sur tous les éléments avec un Iterateur est toujours en O(n) qq soit la collection*	méthode remove() de la classe Iterator<E>	**add(int, E)**	**int indexOf(Object)**	**get(int)**	**E remove(int)**	**int indexOf(Object o)**	**Récupérer un Iterator : ListIterator ListIterator()** *l'itération est elle toujours en O(n)*	**offer(E) (ajout)**	**E peek() (ne supprime pas)**	**E poll()**	**addLast(E)**	**E pollFirst()**	**E first() (ne supprime pas)**	**E last() (ne supprime pas)**	**SortedSet<E> subSet(E fromElement, E toElement)**
ArraList<E>	Collection<E> List<E>	tableau redimensionnable	x			O(1) (ajout à la fin)	O(1)	O(n) ?	O(1)	O(1)	Dans l'ordre des indexs entiers	O(n) ?	O(n)	O(n)	O(n)	O(1)	O(n) ?	O(n)	O(n)								
LinkedList<E>	Collection<E> List<E> Queue<E> Deque<E>	Liste doublement chaînée	x			O(1) (ajout à la fin)	O(n)	O(n)	O(1)	O(1)	Dans l'ordre des indexs entiers	O(1)	O(n)	O(n)	O(n)	O(n)	O(n)	O(n)	O(1)	O(1)	O(1)	O(1)	O(1)				
TreeSet<E>	Collection<E> Set<E> SortedSet<E>	Ensemble ordonné basé sur arbre rouge/ noir (§ABR)	x		x	O(log(n))	O(log(n)) cout amorti	O(log(n)) cout amorti	O(1)	O(1)	Dans l'ordre défini par la relation d'ordre Plus petit d'abord	O(1) cout amorti?												O(1)	O(1)	O(log(n)) ?
HashSet<E>	Collection<E> Set<E> SortedSet<E>	Elements rangés dans une table de hachage	x	x		O(1) cout ammorti	O(1) cout ammorti	O(1) cout ammorti	O(1)	O(1)	Ordre quelconque	O(1)															
PriorityQueue<E>	Collection<E> Queue<E>	Basée sur un tas (heap) : le "plus grand" toujours en tête	x		x	O(log(n))	O(n)	O(log(n))	O(1)	O(1)	Plus petit d'abord ; Ordre quelconque	O(log(n))							O(log(n))	O(1)	O(log(n))					