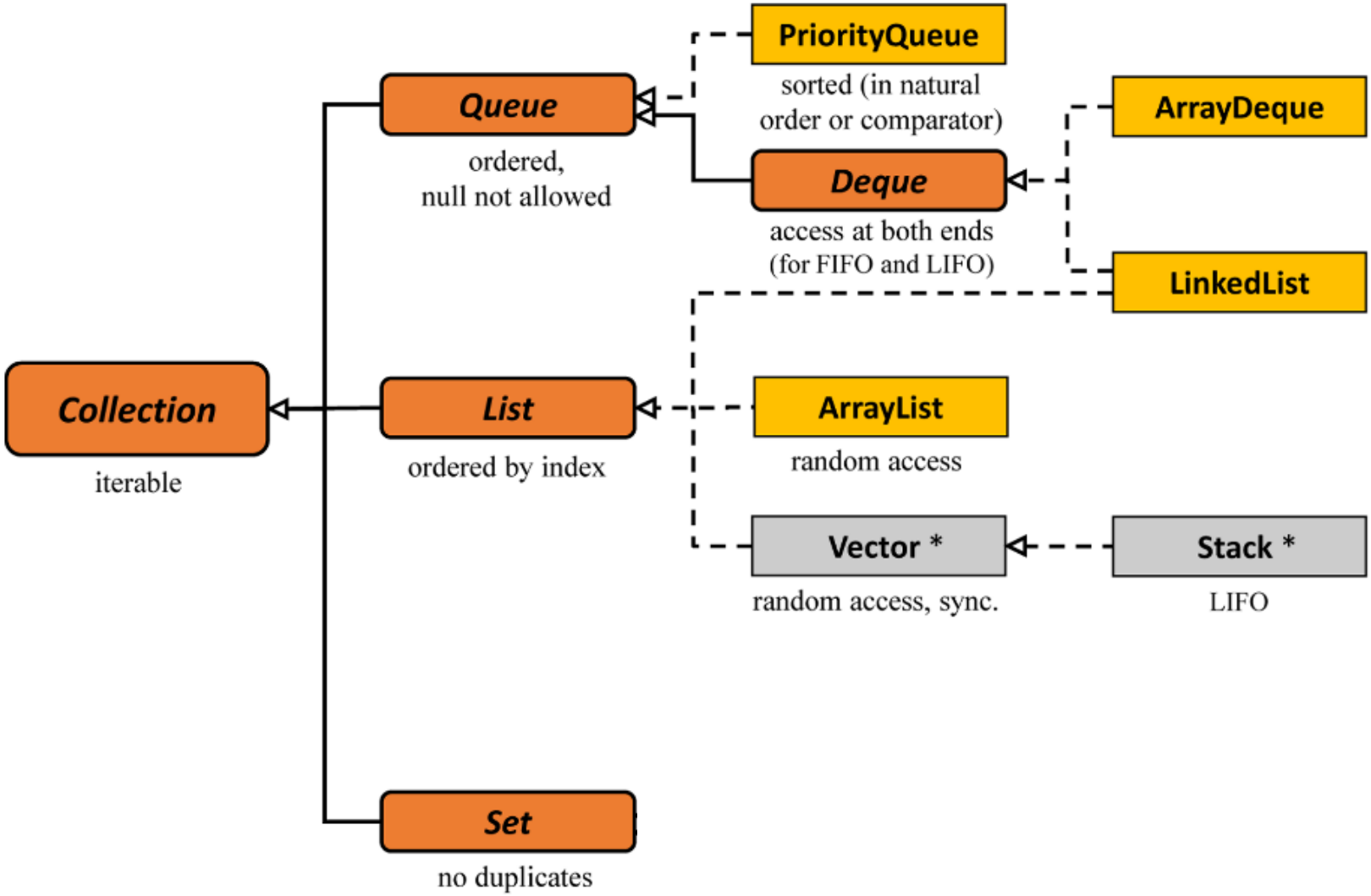
Comparaison des collections Java

Les collections Java



Les listes : ArrayList

- JDK 10 source code:

```
public class ArrayList<E> extends AbstractList<E>
            implements List<E>, RandomAccess, Cloneable,
                       java.io.Serializable
{
     // Attributs
     transient Object[] elementData;
     private   int size;
     ...
}
```

 Tableau redimensionnable

Les listes : LinkedList

- JDK 10 source code:

```
public class LinkedList<E> extends AbstractSequentialList<E>
        implements List<E>, Deque<E>, Cloneable,
                   java.io.Serializable
{
    // Attributs
    transient int size = 0;
    transient Node<E> first;
    transient Node<E> last;
    ...
    // Classe interne Node
    private static class Node<E> {
        E item;
        Node<E> next;
        Node<E> prev;
       ...
    }
}
```

Liste doublement chaînée

ArrayList vs LinkedList

Opération	ArrayList	LinkedList
E get (int index)	O(1)	O(N)
E set (int index, E element)	O(1)	O(N)
boolean add (E element)	O(1) amorti	O(1)
void add (int index, E element)	O(N)	O(N)
E remove (int index)	O(N)	O(N)
boolean contains (Object o)	O(N)	O(N)

- Si nombreux accès en lecture (get) ou écriture (set) → ArrayList
- Si nombreux ajouts (add) ou suppressions (remove) → LinkedList

ArrayList vs LinkedList

ArrayList:

De nombreuses opérations nécessitent N/2 traitements en moyenne (décalages) :

- constante dans le meilleur des cas (fin de liste)
- N dans le pire des cas (début de la liste)

LinkedList:

La plupart des opérations nécessitent N/4 étapes en moyenne (parcours) :

- constante dans le meilleur des cas (par ex. indice=0)
- N/2 dans le pire des cas (milieu de liste)

ArrayList vs LinkedList

ArrayList: ❤️

→ get() et set() en O(1)

LinkedList: ❤️

→ insertions et suppressions via itérateur en O(1)

→ ajouts et retraits en début et en fin de liste en O(1)

ArrayList vs LinkedList

ArrayList:

Mémoire utilisée : capacité

(Capacité initiale par défaut = 10 ; augmentation de 50% si tableau saturé)

LinkedList:

Les pointeurs vers les éléments suivants et précédents sont stockés!

Les deques : ArrayDeque

- JDK 10 source code:

```
public class ArrayDeque<E> extends AbstractCollection<E>
             implements Deque<E>, Cloneable, Serializable
{
    // Attributs
    transient Object[] elements;
    transient int head;
    transient int tail;
    ...
}
```

 Tableau circulaire (sans trou)

Les deques : LinkedList

- JDK 10 source code:

```
public class LinkedList<E> extends AbstractSequentialList<E>
        implements List<E>, Deque<E>, Cloneable,
                   java.io.Serializable
{
    // Attributs
    transient int size = 0;
    transient Node<E> first;
    transient Node<E> last;
    ...
    // Classe interne Node
    private static class Node<E> {
        E item;
        Node<E> next;
        Node<E> prev;
       ...
    }
}
```

Liste doublement chaînée

ArrayDeque vs LinkedList

Opération	ArrayDeque	LinkedList
E getFirst ()	O(1)	O(1)
E getLast ()	O(1)	O(1)
void addFirst (E element)	O(1) amorti	O(1)
void addLast (E element)	O(1) amorti	O(1)
E removeFirst ()	O(1)	O(1)
E removeLast ()	O(1)	O(1)

- Choix par défaut → ArrayDeque
- Si suppressions d'éléments intérieurs à la deque → LinkedList