Class DList<E>

java.lang.Object
    DList<E>

All Implemented Interfaces:

`java.lang.Iterable<E>`

```
public class DList<E>
extends java.lang.Object
implements java.lang.Iterable<E>
```

Constructor Summary

Constructors

Constructor and Description
DList() Set up an empty list by initializing member variables appropriately.
DList(java.util.Comparator<E> comp) Set up an empty list by initializing member variables appropriately.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	**add**(E item) Add item at the end of the list if not sorted OR in the correct location if sorted.
void	**add**(int index, E item) If the list is not sorted, insert the item at index position.

void	**addAll**(int index, java.lang.Iterable<**E**> collection) Insert all elements in the given collection starting at the given index in O(M + N) time where M is the size of current list and N is the size of collection ONLY IF the list is not supposed to be ordered.
void	**clear**() Remove all data (and associated nodes) from this list by setting the head/tail to null.
boolean	**contains**(**E** item) Return true if this list contains item - use equals method for equality check.
boolean	**empty**() Return true if this list is empty, false otherwise.
E	**get**(int index) If index is valid, return the data at index.
int	**indexOf**(**E** item) Return the index of the first occurrence of item.
boolean	**isOrdered**() Return true if this list is sorted (non-null comparator), false otherwise.
java.util.Iterator<**E**>	**iterator**() Return a new Iterator object for this list
int	**lastIndexOf**(**E** item) Return the index of the last occurrence of item.
boolean	**remove**(**E** target) In O(N) time, remove the first occurrence of target in this list
E	**remove**(int index) In O(N) time, if index is valid remove the element at index position in this list and return the removed data
void	**removeRange**(int fromIndex, int toIndex) In O(N) time, remove from this list all of the elements whose index is between fromIndex (inclusive) and toIndex (exclusive).

E	**set**(int index, **E** item) If the list is not sorted and if index is valid, replace the data at index and return the old data that is being replaced.
int	**size**() Return the number of data elements in this list.
java.lang.String	**toString**() Return a print-friendly String representation of this list from front to back.
java.lang.String	**toStringBwd**() Return a print-friendly String representation of this list from back to front (reverse order).

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.lang.Iterable

forEach, spliterator

Constructor Detail

DList

```
public DList()
```

Set up an empty list by initializing member variables appropriately.

DList

```
public DList(java.util.Comparator<E> comp)
```

Set up an empty list by initializing member variables appropriately. This list will main sorted order (in increasing order according to the passed Comparator).

Parameters:

`comp - comparator`

Method Detail

iterator

```
public java.util.Iterator<E> iterator()
```

Return a new Iterator object for this list

Specified by:

`iterator in interface java.lang.Iterable<E>`

toString

```
public java.lang.String toString()
```

Return a print-friendly String representation of this list from front to back. For an empty list return "[ ]" (one space between the opening and closing square brackets); and for a non-empty list return "[d1 d2 d3]"".

Overrides:

`toString in class java.lang.Object`

Returns:

`string representation`

toStringBwd

```
public java.lang.String toStringBwd()
```

Return a print-friendly String representation of this list from back to front (reverse order). For an empty list, return "[ ]" and for a non-empty list return ""[d3 d2 d1]"".

Returns:

`string representation from back to front`

add

```
public void add(E item)
```

Add item at the end of the list if not sorted OR in the correct location if sorted.

Parameters:

item - the item to add

add

```
public void add(int index,
                E item)
```

If the list is not sorted, insert the item at index position. This method should behave the same way as the add(int index, E item) method in the ArrayList class.

Parameters:

index - the position to add

item - the item to add

clear

```
public void clear()
```

Remove all data (and associated nodes) from this list by setting the head/tail to null.

get

```
public E get(int index)
```

If index is valid, return the data at index. Otherwise, return null. 0-based indexing.

Parameters:

index - the index to add

Returns:

data at index

set

```
public E set(int index,
             E item)
```

If the list is not sorted and if index is valid, replace the data at index and return the old data that is being replaced. If the list is sorted and if index is valid, replace the data at index and return the old data only if item is greater than or equal to the data @ prev DNode (if existing) AND smaller than or equal to the

data @ next DNode (if existing). Otherwise, return null. 0-based indexing.

Parameters:

`index - the position to add`

`item - the item to add`

Returns:

`item that is being replaced`

contains

```
public boolean contains(E item)
```

Return true if this list contains item - use equals method for equality check.

Parameters:

`item - the item being checked`

Returns:

`whether or not the list contains the item`

indexOf

```
public int indexOf(E item)
```

Return the index of the first occurrence of item. Return -1 if this list does not contain item. Do NOT use contains method

Parameters:

`item - the item being checked`

Returns:

`the index of the first occurrence of item`

lastIndexOf

```
public int lastIndexOf(E item)
```

Return the index of the last occurrence of item. Return -1 if this list does not contain item. Do NOT use contains method

Parameters:

`item - the item being checked`

Returns:

the index of the last occurrence of item

size

```
public int size()
```

Return the number of data elements in this list.

Returns:

the number of data elements in this list

isOrdered

```
public boolean isOrdered()
```

Return true if this list is sorted (non-null comparator), false otherwise.

Returns:

whether or not the list is sorted

empty

```
public boolean empty()
```

Return true if this list is empty, false otherwise.

Returns:

whether or not the list is empty

remove

```
public E remove(int index)
```

In O(N) time, if index is valid remove the element at index position in this list and return the removed data

Parameters:

`index - the position to remove`

Returns:

`removed data`

Throws:

`java.lang.IndexOutOfBoundsException - if index is invalid`

remove

```
public boolean remove(E target)
```

In O(N) time, remove the first occurrence of target in this list

Parameters:

`target - the item to be removed`

Returns:

`whether or not the target has been found`

removeRange

```
public void removeRange(int fromIndex,
                        int toIndex)
```

In O(N) time, remove from this list all of the elements whose index is between fromIndex (inclusive) and toIndex (exclusive). If fromIndex is too small (< 0) remove from the beginning of the list and if toIndex is too large (> size)remove until the end of the list.

Parameters:

`fromIndex - starting index (inclusive)`

`toIndex - ending index (exclusive`

Throws:

`java.lang.IndexOutOfBoundsException - if the list is empty (regardless of indices)`

java.lang.IllegalArgumentException - if the indices are out of order (i.e., fromIndex > toIndex)

addAll

```
public void addAll(int index,
                   java.lang.Iterable<E> collection)
```

Insert all elements in the given collection starting at the given index in O(M + N) time where M is the size of current list and N is the size of collection ONLY IF the list is not supposed to be ordered. Use an iterator to traverse the given collection. Similiar to add(int, E), allow user to add to the end of the list

Parameters:

index - position to add elements

collection - elements to be inserted

Throws:

java.lang.NullPointerException - if collection is null

java.lang.IndexOutOfBoundsException - if the specified index is invalid