java.util

**Class LinkedList<E>**

* [java.lang.Object](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html" \o "class in java.lang)
* [java.util.AbstractCollection](https://docs.oracle.com/javase/7/docs/api/java/util/AbstractCollection.html)<E>
  + [java.util.AbstractList](https://docs.oracle.com/javase/7/docs/api/java/util/AbstractList.html)<E>
    - [java.util.AbstractSequentialList](https://docs.oracle.com/javase/7/docs/api/java/util/AbstractSequentialList.html)<E>
      * java.util.LinkedList<E>
* **Type Parameters:**

E - the type of elements held in this collection

**All Implemented Interfaces:**

[Serializable](https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html), [Cloneable](https://docs.oracle.com/javase/7/docs/api/java/lang/Cloneable.html), [Iterable](https://docs.oracle.com/javase/7/docs/api/java/lang/Iterable.html)<E>, [Collection](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)<E>, [Deque](https://docs.oracle.com/javase/7/docs/api/java/util/Deque.html)<E>, [List](https://docs.oracle.com/javase/7/docs/api/java/util/List.html)<E>, [Queue](https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html)<E>

public class **LinkedList<E>**

extends [AbstractSequentialList](https://docs.oracle.com/javase/7/docs/api/java/util/AbstractSequentialList.html)<E>

implements [List](https://docs.oracle.com/javase/7/docs/api/java/util/List.html)<E>, [Deque](https://docs.oracle.com/javase/7/docs/api/java/util/Deque.html)<E>, [Cloneable](https://docs.oracle.com/javase/7/docs/api/java/lang/Cloneable.html), [Serializable](https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html)

Doubly-linked list implementation of the List and Deque interfaces. Implements all optional list operations, and permits all elements (including null).

All of the operations perform as could be expected for a doubly-linked list. Operations that index into the list will traverse the list from the beginning or the end, whichever is closer to the specified index.

**Note that this implementation is not synchronized.** If multiple threads access a linked list concurrently, and at least one of the threads modifies the list structurally, it *must* be synchronized externally. (A structural modification is any operation that adds or deletes one or more elements; merely setting the value of an element is not a structural modification.) This is typically accomplished by synchronizing on some object that naturally encapsulates the list. If no such object exists, the list should be "wrapped" using the [Collections.synchronizedList](https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html" \l "synchronizedList(java.util.List)) method. This is best done at creation time, to prevent accidental unsynchronized access to the list:

List list = Collections.synchronizedList(new LinkedList(...));

The iterators returned by this class's iterator and listIterator methods are *fail-fast*: if the list is structurally modified at any time after the iterator is created, in any way except through the Iterator's own remove or add methods, the iterator will throw a [ConcurrentModificationException](https://docs.oracle.com/javase/7/docs/api/java/util/ConcurrentModificationException.html" \o "class in java.util). Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Note that the fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw ConcurrentModificationException on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: *the fail-fast behavior of iterators should be used only to detect bugs.*