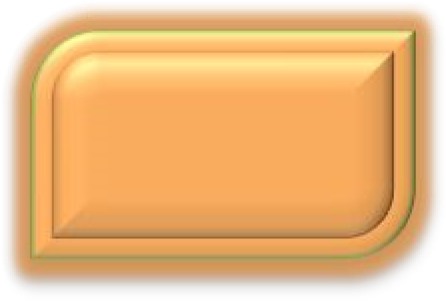
**JAC444 - Lecture 9**

Java Collections Segment 3 - List

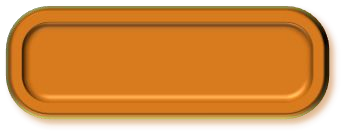
# Type Hierarchy



Interface



Abstract



LinkedList



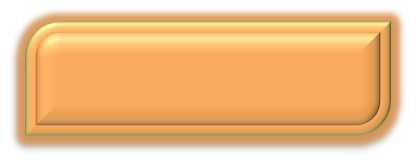
Iterable



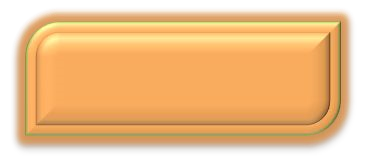
Collection



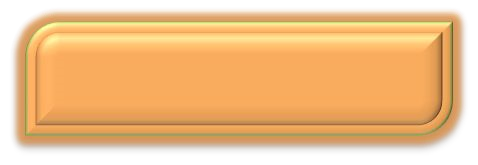
List



AbstractCollection



AbstractList



AbstractSequentialList



ArrayList



Vector



Class

# The List<E> Interface

A **List** is an ordered **Collection** (called a sequence)

**public interface List<E> extends Collection<E> {**

**// Positional Access**

**E get(int index);**

**E set(int index, E element); 1** Access  **void add(int index, E element);**

**E remove(int index); boolean addAll(int index, Collection c);**

**// Search**

**int indexOf(E o); 2** Search  **int lastIndexOf(E o);**

**// Iteration**

**ListIterator listIterator(); 3** Iteration

**ListIterator listIterator(int index);**

**// Range-view**

**List subList(int from, int to); 4** Range

**}**

**ListIterator<E>**

|  |
| --- |
| **public interface ListIterator<E> extends Iterator<E> {**  **boolean hasNext(); E next();**  **boolean hasPrevious(); E previous();**  **int nextIndex(); int previousIndex();**  **void remove(); // Optional void set(E o); // Optional void add(E o); // Optional**  **}** |

Standard idiom for iterating backwards through a list

**for ( ListIterator<E> i = list.listIterator(list.size()); i.hasPrevious();) { E o = i.previous();**

# Operations on List

* Positional access — manipulates elements based on their numerical position in the list.
* Search — a specified object in the list and returns its numerical position.
* Iteration — extends Iterator semantics to take advantage of the list's sequential nature.

* Range-view — The sublist method performs arbitrary range operations on the list.

# Cursor Positions in a List

The cursor is always between two elements of a list

Element 1

Element 2

Element 3

Element 4

0

1

2

3

4

**next**

**previous**

In a list of length n, there are n+1 valid values for index, from 0 to n, inclusive.

# List Implementations

There are two List implementations:

1. **ArrayList<E>** fast, offers constant-time positional access
2. **LinkedList<E>** better for adding elements at the

beginning or deleting from interior

**LinkedList**

* The **LinkedList<E>** class extends **AbstractSequentialList<E>** and implements the **List<E>** interface
* It has two constructors: the default and

**LinkedList(Collection<? extends E> c)**

* Multithreaded

**Collections.synchronizedList(new LinkedList(...));**

* Important method **public <T> T[] toArray(T[] a)**

Returns an array containing all of the elements in this list in proper sequence

# Working with List

new List() does not work, since List is an interface

To build an object of type List one needs to use the implementations without exposing the implementation Idioms:

List<E> list = new LinkedList<E>();

List<E> list = new ArrayList<E>(); Never expose the implementation:

ArrayList<E> list = new ArrayList<E>();

# Position Access

Example to swap list elements at positions k and h:

**public <String> void swap(List<String> list, int k, int h) { String s = list.get(k); list.set(k, list.get(h)); list.set(h, s);**

**}**

# Algorithms on List

Some of the important algorithms on list:

|  |  |
| --- | --- |
| **sort** | Sorts list using mergesort |
| **shuffle** | Randomly shuffles elements |
| **reverse** | Reverses order of elements |
| **rotate** | Rotates list by specified number |
| **fill** | Overwrites every element with specified element |
| **copy** | Copies source list into destination |
| **binarySearch** | Performs search using binary search |