# Iterators and List Iterators

Sometimes we need to access every item in a data structure or collection

- We call this ***traversing*** or ***iterating over*** the data structure or collection
- E.g:

```
for (int i = 0; i < arr.length(); i++)
        /*  loop body*/
```

- What if we want to traverse a ***collection*** of objects? we may or may not know its underlying implementation.
- Java provides a common scheme for stepping through all elements in *any* collection, called an **iterator**
- Java provides an `Iterator<T>` interface
  - Any object of any class that satisfies the **`Iterator<T>`** interface is an **`Iterator<T>`**

# Iterators

- An iterator is an object that is used with a collection to provide sequential access to the collection elements
  - This access allows examination and possible modification of the elements

# Iterator Methods

| Modifier and Type | Method and Description |
| --- | --- |
| `default void` | **`forEachRemaining`**`(Consumer<? super E> action)`<br>Performs the given action for each remaining element until all elements have been processed or the action throws an exception. |
| `boolean` | **`hasNext`**`()`<br>Returns `true` if the iteration has more elements. |
| **`E`** | **`next`**`()`<br>Returns the next element in the iteration. |
| `default void` | **`remove`**`()`<br>Removes from the underlying collection the last element returned by this iterator (optional operation). |

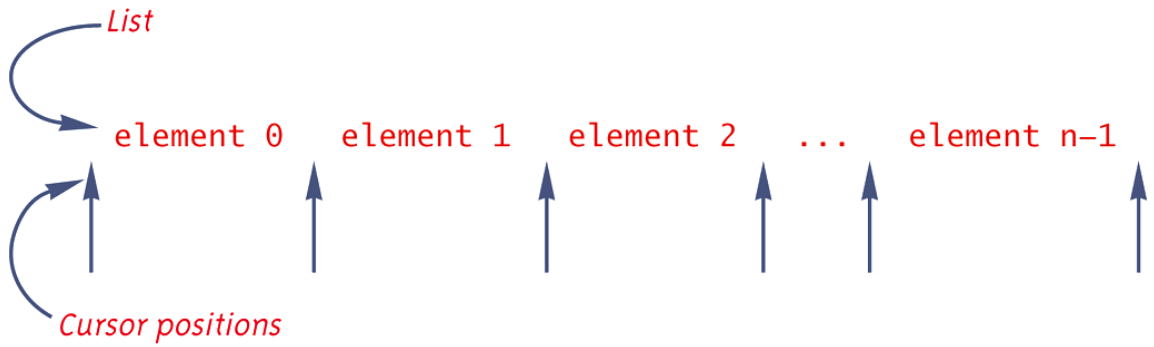
https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html

# List- Iterator

- An iterator for lists that allows the programmer to traverse the list in either direction, modify the list during iteration, and obtain the iterator's current position in the list. A ListIterator has no current element; its *cursor position* always lies between the element that would be returned by a call to previous() and the element that would be returned by a call to next().

# List Iterator : Cursor Positions

**Display 16.11** `ListIterator<T>` **Cursor Positions**




*The default initial cursor position is the leftmost one.*

# Iterator vs. ListIterator

- Using ListIterator You can
  - iterate backwards
  - obtain the iterator at any point.
  - add a new value at any point.
  - set a new value at that point

# Methods

| Modifier and Type | Method and Description |
| --- | --- |
| void | **add**(**E** e)<br>Inserts the specified element into the list (optional operation). |
| boolean | **hasNext**()<br>Returns `true` if this list iterator has more elements when traversing the list in the forward direction. |
| boolean | **hasPrevious**()<br>Returns `true` if this list iterator has more elements when traversing the list in the reverse direction. |
| **E** | **next**()<br>Returns the next element in the list and advances the cursor position. |
| int | **nextIndex**()<br>Returns the index of the element that would be returned by a subsequent call to **next()**. |
| **E** | **previous**()<br>Returns the previous element in the list and moves the cursor position backwards. |
| int | **previousIndex**()<br>Returns the index of the element that would be returned by a subsequent call to **previous()**. |
| void | **remove**()<br>Removes from the list the last element that was returned by **next()** or **previous()** (optional operation). |
| void | **set**(**E** e)<br>Replaces the last element returned by **next()** or **previous()** with the specified element (optional operation). |

# Example

- Lets see an example from:
  - http://www.tutorialspoint.com/java/java_using_iterator.htm

# Length and the capacity of an arrayList in Java

- While it is not very explicitly documented, a careful examination of the online material that discusses arrays in Java makes it clear that the length of an array in Java must be small enough to be represented as an int.
- Furthermore, the fact that an ArrayList is almost certainly implemented using an underlying static array suggests that the capacity of an ArrayList must be this small, as well.
- In other words (according to the textbook) neither the length of an array nor the capacity of an ArrayList can be greater than

  2,147,483,647

- so neither an array nor an ArrayList can be used to store more than couple of billion entries!

- Describe a way to show how, “in principle,” you could modify this restriction in order to support arrays that store as many as
  - 2,147,483,647 × 2,147,483,647

- Given the java programs in the tutorial page, try to get information about the time needed to run each of these programs.
  - time java declareLargeArray1

- If you failed to run some of the programs try to use :
  - java -Xmx3500m declareLargeArray6

- (run Java with more memory available)

- That said, you *should* have noticed that, after a while, the time used was growing **more than linearly** in the length of the array being declared (if, indeed, you were able to have programs executed at all) — it is possible that you noticed a ***significant*** increase in the times needed to initialize two arrays of “similar” sizes.