JAVA - HOW TO USE ITERATOR?

Previous Page
Next Page

Often, you will want to cycle through the elements in a collection. For example, you might want to display each element.

The easiest way to do this is to employ an iterator, which is an object that implements either the Iterator or the ListIterator interface.

Iterator enables you to cycle through a collection, obtaining or removing elements. ListIterator extends Iterator to allow bidirectional traversal of a list, and the modification of elements.

Before you can access a collection through an iterator, you must obtain one. Each of the collection classes provides an iterator method that returns an iterator to the start of the collection. By using this iterator object, you can access each element in the collection, one element at a time.

In general, to use an iterator to cycle through the contents of a collection, follow these steps:

- Obtain an iterator to the start of the collection by calling the collection's iterator method.
- Set up a loop that makes a call to hasNext. Have the loop iterate as long as hasNext returns true.
- Within the loop, obtain each element by calling next.

For collections that implement List, you can also obtain an iterator by calling ListIterator.

The Methods Declared by Iterator:

SN	Methods with Description
1	**boolean hasNext** Returns true if there are more elements. Otherwise, returns false.
2	**Object next** Returns the next element. Throws NoSuchElementException if there is not a next element.
3	**void remove** Removes the current element. Throws IllegalStateException if an attempt is made to call remove that is not preceded by a call to next.

The Methods Declared by ListIterator:

SN	Methods with Description
1	**void add*Objectobj*** Inserts obj into the list in front of the element that will be returned by the next call to next.
2	**boolean hasNext** Returns true if there is a next element. Otherwise, returns false.

3 **boolean hasPrevious**

Returns true if there is a previous element. Otherwise, returns false.

4 **Object next**

Returns the next element. A NoSuchElementException is thrown if there is not a next element.

5 **int nextIndex**

Returns the index of the next element. If there is not a next element, returns the size of the list.

6 **Object previous**

Returns the previous element. A NoSuchElementException is thrown if there is not a previous element.

7 **int previousIndex**

Returns the index of the previous element. If there is not a previous element, returns -1.

8 **void remove**

Removes the current element from the list. An IllegalStateException is thrown if remove is called before next or previous is invoked.

9 **void set*Objectobj***

Assigns obj to the current element. This is the element last returned by a call to either next or previous.

Example:

Here is an example demonstrating both Iterator and ListIterator. It uses an ArrayList object, but the general principles apply to any type of collection.

Of course, ListIterator is available only to those collections that implement the List interface.

```
import java.util.*;

public class IteratorDemo {

   public static void main(String args[]) {
      // Create an array list
      ArrayList al = new ArrayList();
      // add elements to the array list
      al.add("C");
      al.add("A");
      al.add("E");
      al.add("B");
      al.add("D");
      al.add("F");

      // Use iterator to display contents of al
      System.out.print("Original contents of al: ");
      Iterator itr = al.iterator();
      while(itr.hasNext()) {
```

```
         Object element = itr.next();
         System.out.print(element + " ");
      }
      System.out.println();

   // Modify objects being iterated
      ListIterator litr = al.listIterator();
      while(litr.hasNext()) {
         Object element = litr.next();
         litr.set(element + "+");
      }
      System.out.print("Modified contents of al: ");
      itr = al.iterator();
      while(itr.hasNext()) {
         Object element = itr.next();
         System.out.print(element + " ");
      }
      System.out.println();

      // Now, display the list backwards
      System.out.print("Modified list backwards: ");
      while(litr.hasPrevious()) {
         Object element = litr.previous();
         System.out.print(element + " ");
       }
       System.out.println();
    }
}
```

This would produce the following result:

```
Original contents of al: C A E B D F
odified contents of al: C+ A+ E+ B+ D+ F+
odified list backwards: F+ D+ B+ E+ A+ C+
```