

Cursors

18 February 2025 18:37

There are basically 3 types of cursors:

- 1) Enumeration
- 2) Iterator
- 3) ListIterator

If you want to get objects one by one from the Collection then we should go for Cursors.

1. Enumeration: we can use enumeration to get objects one-by-one from legacy collection object. We can create enumeration object by using elements() method of vector class.

```
Enumeration e = v.elements();
```

Methods: public boolean hasMoreElements()
public Object nextElement()

These two methods of
Enumeration is implemented
in StringTokenizer class.

Interface Enumeration<E>

All Known Subinterfaces:

NamingEnumeration<T>

All Known Implementing Classes:

StringTokenizer

public interface Enumeration<E>

An object that implements the Enumeration interface generates a series of elements, one at a time. Successive calls to the nextElement method return successive elements of the series.

For example, to print all elements of a Vector<E> v:

```
for (Enumeration<E> e = v.elements(); e.hasMoreElements();  
System.out.println(e.nextElement());
```

Methods are provided to enumerate through the elements of a vector, the keys of a hashtable, and the values in a hashtable. Enumerations are also used to specify the input streams to a SequenceInputStream.

NOTE: The functionality of this interface is duplicated by the Iterator interface. In addition, Iterator adds an optional remove operation, and has shorter method names. New implementations should consider using Iterator in preference to Enumeration.

Since:
JDK1.0

Method Summary

All Methods	Instance Methods	Abstract Methods
Modifier and Type	Method and Description	
boolean	hasMoreElements()	Tests if this enumeration contains more elements.
E	nextElement()	Returns the next element of this enumeration if this enumeration object has at least one more element to provide.

```
package Chapter2;  
import java.util.Enumeration;  
import java.util.Vector;  
public class EnumerationDemo {  
    public static void main(String[] args) {  
        Vector v = new Vector();  
        for(int i = 0; i <=10; i++) {  
            v.addElement(i);  
        }  
        System.out.println(v);  
        System.out.println("Let's enumerate:");  
        for(Enumeration<Integer> e = v.elements(); e.hasMoreElements();){  
            Integer value = (Integer)e.nextElement();  
            System.out.println(value);  
        }  
    }  
}
```

Limitations of Enumeration

1. We can apply enumeration concept only for legacy classes. And it is not a universal cursor. And it is not a universal cursor.
2. By using enumeration, we can get only read access. And we cannot perform remove operation.
3. In enumeration, we can move in 1DIRECTION only, that is forward direction.

Iterator interface

We can apply iterator concept for any collection object. And hence, it is a universal cursor.

By using iterator, we can perform both read and remove operations.

We can create Iterator object using iterator() method.

Iterator iterator(); of Collection interface

Iterator itr = c.iterator();

↑
Any collection object

1. public boolean hasNext();
2. public Object next();
3. public void remove();

Interface Iterator<E>

Type Parameters:

E - the type of elements returned by this iterator

All Known Subinterfaces:

ListIterator<E>, PrimitiveIterator<T,T_CONS>, PrimitiveIterator.OfDouble, PrimitiveIterator.OfInt, PrimitiveIterator.OfLong, XMLEventReader

All Known Implementing Classes:

BeanContextSupport.BCSIterator, EventReaderDelegate, Scanner

public interface Iterator<E>

An iterator over a collection. Iterator takes the place of Enumeration in the Java Collections Framework. Iterators differ from enumerations in two ways:

- Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
- Method names have been improved.

This interface is a member of the Java Collections Framework.

Since:

1.2

See Also:

Collection, ListIterator, Iterable

Method Summary

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

Very useful method for iterators in Java.

<code>Iterator<E></code>	<code>iterator()</code> Returns an iterator over the elements in this list in proper sequence.
<code>int</code>	<code>lastIndexOf(Object o)</code> Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
<code>ListIterator<E></code>	<code>listIterator()</code> Returns a list iterator over the elements in this list (in proper sequence).
<code>ListIterator<E></code>	<code>listIterator(int index)</code> Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.

```

package Chapter2;
import java.util.*;

public class IteratorDemo {
    public static void main(String[] args) {
        ArrayList al = new ArrayList();
        for(int i = 0; i <= 10; i++) {
            al.add(i);
        }
        System.out.println("ArrayList: " + al);
        Iterator iter = al.iterator();
        while (iter.hasNext()) {
            Integer i = (Integer)iter.next();
            if(i % 2 == 0) {
                System.out.println(i);
            }
            else {
                iter.remove();
            }
        }
        System.out.println("Even arraylist: " + al);
    }
}

```

Output:

```

ArrayList: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
0
2
4
6
8
10
Even arraylist: [0, 2, 4, 6, 8, 10]

```

Limitations of Iterator

- Forward direction movement is applicable only we cannot move in backward direction. Enumeration and iterator both are single direction cursor.
- By using iterator, we can perform only read and remove operation. And we cannot perform replacement and addition of new objects.

To overcome the second limitation of iterator ListIterator interface is developed.

ListIterator

1. By using list iterator, we can move either to the forward direction or to the backward direction And hence, it is a bidirectional cursor.
2. By using list iterator, we can perform replacement and addition of new objects in addition to read and remove operation.

`public ListIterator listIterator();`

`ListIterator liter = obj.listIterator();`

Any collection object which has

`Iterator(I)`

`ListIterator(I)`

ListIterator(I)

Any collection object which has List properties.

ListIterator is a child interface of Iterator interface. And hence, all methods present in the iterator interface by default available to the ListIterator.

Public boolean hasNext() } Forward movement
public Object next() } Delete
public int nextIndex() } CRUD operations
public boolean hasPrevious() } Create
public Object previous() } Read
public int previousIndex() } Update

public void remove(); } Backward movement.
public void add (Object ob) } delete, add, modify or update.
public void set (Object ob) } replaces the current object with ob.

```
package Chapter2;
import java.util.*;
public class ListIteratorDemo {
    public static void main(String[] args) {
        LinkedList liList = new LinkedList();
        liList.add("BalaKrishna");
        liList.add("Rama");
        liList.add("Ravichandran");
        liList.add("LakshmiNarayan");
        liList.add("Bharat");
        System.out.println(liList);
        ListIterator liter = liList.listIterator();
        while (liter.hasNext()) {
            String str = (String)liter.next();
            if(str.equals("Rama")){
                liter.remove();
            }
            else if(str.equals("Bharat")){
                liter.add("Lakshman");
            }
            else if(str.equals("Ravichandran")){
                liter.set("RC");
            }
        }
        System.out.println("List: " + liList);
        while(liter.hasPrevious()){
            String str = (String)liter.previous();
            System.out.println(str);
        }
    }
}
```

```
}
```

Output:

```
[BalaKrishna, Rama, Ravichandran, LakshmiNarayan, Bharat]
List: [BalaKrishna, RC, LakshmiNarayan, Bharat, Lakshman]
Lakshman
Bharat
LakshmiNarayan
RC
BalaKrishna
```

Note:- The most powerful cursor is ListIterator but its limitation is, it is applicable only for List Objects.

Comparison Table of 3 cursors

Properties	Enumeration	Iterator	List Iterator
Where we can Apply?	Only applicable for legacy classes.	For any Collection Object.	Only for List Objects.
Is it legacy?	Yes (1.0 Version)	No (1.2 Version)	No (1.2 Version)
Movement	Single direction(only forward direction)	Single direction(only forward direction)	Bi-directional
Allowed Operations	Only Read.	Read and remove.	Read, remove, replace, and add new objects.
How we can get object?	By using elements() method of vector class.	By using iterator() of Collection interface.	By using iterator() of List interface.
Methods	2-Methods 1. hasMoreElements() 2. nextElement()	3-methods 1. hasNext() 2. next() 3. remove()	9-methods 1. hasNext() 2. next() 3. nextIndex() 4. hasPrevious() 5. previousIndex() 6. previous() 7. remove() 8. add(Object o) 9. set(Object o)

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

Internal implementation of Cursor

```

package Chapter2;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.ListIterator;
import java.util.Vector;
public class InternalClassOfCursor {
    public static void main(String[] args) {
        Vector v = new Vector();
        Enumeration e = v.elements();
        Iterator itr = v.listIterator();
        ListIterator litr = v.listIterator();
        System.out.println("\f"+e.getClass().getName());
        System.out.println("\n"+itr.getClass().getName());
        System.out.println("\n"+litr.getClass().getName());
    }
}

```

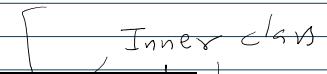
anonymous inner class

Output:

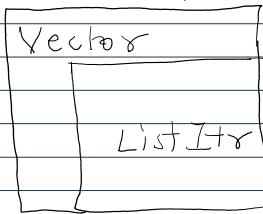
```

java.util.Vector$1
java.util.Vector$ListItr
java.util.Vector$ListItr

```



Vector class has implemented all the methods of Enumeration



Iterators & ListIterator
methods are implemented.