Retrieving Elements from Collection in Java (For-each, Iterator, ListIterator & EnumerationIterator)

Prerequisite : Collection in Java

Following are the 4 ways to retrieve any elements from a collection object:

For-each

For each loop is meant for traversing items in a collection.

```
// Iterating over collection 'c' using for-each
   for (Element e: c)
       System.out.println(e);
```

We read the ':' used in for-each loop as "in". So loop reads as "for each element e in elements", here elements is the collection which stores Element type items.

Note : In Java 8 using lambda expressions we can simply replace for-each loop with

```
elements.forEach (e -> System.out.println(e) );
```

Using Cursors

Cursor is an interface and it is used to retrieve data from collection object,one by one. Cursor has 3 types, which are given below:

1. **Iterator Interface**: Iterator is an interface provided by collection framework to traverse a collection and for a sequential access of items in the collection.

```
// Iterating over collection 'c' using terator
for (Iterator i = c.iterator(); i.hasNext(); )
    System.out.println(i.next());
```

It has 3 methods:

- ***boolean hasNext():*** This method returns true if the iterator has more elements.
- ***elements next()***: This method returns the next elements in the iterator.
- ***void remove()***: This method removes from the collection the last elements returned by the iterator.

```
// Java program to demonstrate working of iterators
import java.util.*;
public class IteratorDemo
{
    public static void main(String args[])
    {
        //create a Hashset to store strings
        HashSet<String> hs = new HashSet<String>() ;

        // store some string elements
        hs.add("India");
        hs.add ("America");
        hs.add("Japan");

        // Add an Iterator to hs.
        Iterator it = hs.iterator();

        // Display element by element using Iterator
        while (it.hasNext())
            System.out.println(it.next());
    }
}
```

Run on IDE

Output:

```
America
Japan
India
```

Refer Iterator vs For-each for differences between iterator and for-each.

2. **ListIterator Interface**: It is an interface that contains methods to retrieve the elements from a collection object, both in **forward and reverse directions**. This iterator is for list based collections. It has following important methods:

   - ***booleanhasNext()***: This returns true if the ListIterator has more elements when traversing the list in the forward direction.
   - ***booleanhasPerivous()***: This returns true if the ListIterator has more elements when traversing the list in the reverse direction.
   - ***element next()***: This returns the next element in the list.
   - ***element previous()***:This returns the previous element in the list.

- ***void remove()***: This removes from the list the last elements that was returned by the next() or previous() methods.
- ***int nextIndex()*** Returns the index of the element that would be returned by a subsequent call to next(). (Returns list size if the list iterator is at the end of the list.)
- ***int previousIndex()*** Returns the index of the element that would be returned by a subsequent call to previous(). (Returns -1 if the list iterator is at the beginning of the list.)

Source: ListIterator Oracle Docs

How is Iterator different from ListIterator?

Iterator can retrieve the elements only in forward direction. But ListIterator can retrieve the elements in forward and reverse direction also. So ListIterator is preferred to Iterator.

Using ListItarator, we can get iterator's current position

Since ListIterator can access elements in both directions and supports additional operators, ListIterator cannot be applied on Set (e.g., HashSet and TreeSet. See this). However, we can use LisIterator with vector and list (e.g. ArrayList).

```
// Java program to demonstrate working of ListIterator
import java. util.* ;

class Test
{
    public static void main(String args[])
    {
        // take a vector to store Integer objects
        Vector<Integer> v = new Vector<Integer>();

        // Adding Elements to Vector
        v.add(10);
        v.add(20);
        v.add(30);

        // Creating a ListIterator
        ListIterator lit = v.listIterator();
        System.out.println("In Forward direction:");

        while (lit.hasNext())
            System.out.print(lit.next()+" ") ;

        System.out.print("\n\nIn backward direction:\n") ;
        while (lit.hasPrevious())
            System.out.print(lit.previous()+" ");
    }
}
```

Run on IDE

Output :

```
In Forward direction:
10 20 30
```

```
In backward direction:
30 20 10
```

3. **EnumerationIterator Interface**: The interface is useful to retrieve one by one the element. This iterator is based on data from Enumeration and has methods:

   - ***booleanhasMoreElements()***: This method tests if the Enumeration has any more elements or not .
   - ***element nextElement()***: This returns the next element that is available in elements that is available in Enumeration

   What is the difference between Iterator and Enumeration?

   Iterator has an option to remove elements from the collection which is not available in Enumeration. Also, Iterator has methods whose names are easy to follow and Enumeration methods are difficult to remember.

```
import java.util.Vector;
import java.util.Enumeration;
public class Test
{
    public static void main(String args[])
    {
        Vector dayNames = new Vector();
        dayNames.add("Sunday");
        dayNames.add("Monday");
        dayNames.add("Tuesday");
        dayNames.add("Wednesday");
        dayNames.add("Thursday");
        dayNames.add("Friday");
        dayNames.add("Saturday");

        // Creating enumeration
        Enumeration days = dayNames.elements();

        // Retrieving elements of enumeration
        while (days.hasMoreElements())
            System.out.println(days.nextElement());
    }
}
```

Run on IDE

Output:

```
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
```