Accessing a Java Collection using Iterators

To access elements of a collection, either we can use index if collection is list based or we need to traverse the element. There are multiple ways to traverse through the elements of any collection.

1) Using Iterator interface

2) Using ListIterator interface

- **Accessing elements using Iterator**

Iterator is an interface that is used to iterate the collection elements. It is part of java collection framework. It provides some methods that are used to check and access elements of a collection.

Iterator Interface is used to traverse a list in forward direction, enabling you to remove or modify the elements of the collection. Each collection classes provide iterator() method to return an iterator.

- **Iterator Interface Methods**

Method	**Description**
boolean hasNext()	Returns **true** if there are more elements in the collection. Otherwise, returns false.
E next()	Returns the **next element present** in the collection. Throws NoSuchElementException if there is not a next element.
void remove()	Removes the current element. Throws IllegalStateException if an attempt is made to call remove() method that is not preceded by a call to next() method.

- **Iterator Example**

In this example, we are using iterator() method of collection interface that returns an instance of Iterator interface. After that we are using hasNext() method that returns true of collection contains an elements and within the loop, obtain each element by calling next() method.

```
import java.util.Iterator;
class Demo
{
  public static void main(String[] args)
  {
    ArrayList< String> ar = new ArrayList< String>();
    ar.add("ab");
    ar.add("bc");
    ar.add("cd");
    ar.add("de");
    Iterator it = ar.iterator();    //Declaring Iterator
    while(it.hasNext())
    {
      System.out.print(it.next()+" ");
    }
  }
}
```

Output

```
ab bc cd de
```

Use an iterator to remove numbers less than 10 from a collection:

```
import java.util.ArrayList;
import java.util.Iterator;
public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> numbers = new ArrayList<Integer>();
    numbers.add(12);
    numbers.add(8);
    numbers.add(2);
    numbers.add(23);
    Iterator<Integer> it = numbers.iterator();
    while(it.hasNext()) {
      Integer i = it.next();
      if(i < 10) {
        it.remove();
      }
    }
    System.out.println(numbers);
  }
}
```

- **Accessing elements using ListIterator**

It is only applicable for List collection implemented classes like ArrayList, LinkedList, etc. It provides bi-directional iteration. ListIterator must be used when we want to enumerate elements of List. This cursor has more functionality(methods) than iterator. ListIterator object can be created by calling listIterator() method present in the List interface.

Syntax:

ListIterator ltr = l.listIterator();

All three methods of Iterator interface are available for ListIterator. In addition, there are **six** more methods.

1. Forward direction

1.1 hasNext(): Returns true if the iteration has more elements

public boolean hasNext();

1.2 next(): Same as next() method of Iterator. Returns the next element in the iteration.

public Object next();

1.3 nextIndex(): Returns the next element index or list size if the list iterator is at the end of the list.

public int nextIndex();

2. Backward direction

2.1 hasPrevious(): Returns true if the iteration has more elements while traversing backward.

public boolean hasPrevious();

2.2 previous(): Returns the previous element in the iteration and can throw **NoSuchElementException** if no more element present.

public Object previous();

2.3 previousIndex(): Returns the previous element index or -1 if the list iterator is at the beginning of the list,

public int previousIndex();

3. Other Methods

3.1 remove(): Same as remove() method of Iterator. Removes the next element in the iteration.

public void remove();

3.2 set(Object obj): Replaces the last element returned by next() or previous() with the specified element.

public void set(Object obj);

3.3 add(Object obj): Inserts the specified element into the list at the position before the element that would be returned by next()

public void add(Object obj);

ListIterator Example:

```
import java.util.ArrayList;
import java.util.ListIterator;
public class Main{
   public static void main(String[] args) {
      ArrayList al = new ArrayList();

      for (int i = 0; i < 10; i++)
         al.add(i);
      System.out.println(al);

      ListIterator ltr = al.listIterator();
```

```
        while (ltr.hasNext()) {
           int i = (Integer)ltr.next();

           System.out.print(i + " ");

           if (i % 2 == 0) {
              i++;
              ltr.set(i);
              ltr.add(i);
           }
        }

        System.out.println();
        System.out.println(al);
    }
}
```

Important Points

- Please note that initially, any iterator reference will point to the index just before the index of the first element in a collection.

- We don’t create objects of Iterator, ListIterator because they are interfaces. We use methods like iterator(), listIterator() to create objects. These methods have an anonymous Inner Class that extends respective interfaces and return this class object.