ListIterator in Java with examples

BY CHAITANYA SINGH | FILED UNDER: JAVA COLLECTIONS

In the last tutorial, we discussed Iterator in Java using which we can traverse a List or Set in forward direction. Here we will discuss ListIterator that allows us to traverse the list in both directions (forward and backward).

ListIterator Example

In this example we are traversing an ArrayList in both the directions.

```
import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;

public class ListIteratorExample {
  public static void main(String a[]){
    ListIterator<String> litr = null;
    List<String> names = new ArrayList<String>();
    names.add("Shyam");
    names.add("Rajat");
    names.add("Paul");
    names.add("Tom");
    names.add("Kate");
    //Obtaining list iterator
    litr=names.listIterator();

    System.out.println("Traversing the list in forward direction:");
    while(litr.hasNext()){
       System.out.println(litr.next());
    }
    System.out.println("\nTraversing the list in backward direction:");
    while(litr.hasPrevious()){
       System.out.println(litr.previous());
    }
  }
}
```

Output:

```
Traversing the list in forward direction:
Shyam
Rajat
Paul
Tom
Kate

Traversing the list in backward direction:
Kate
Tom
Paul
Rajat
Shyam
```

Note: We can use Iterator to traverse List and Set both but using ListIterator we can only traverse list. There are several other differences between Iterator and ListIterator, we will discuss them in next post.

Methods of ListIterator

1) void add(E e): Inserts the specified element into the list (optional operation).
2) boolean hasNext(): Returns true if this list iterator has more elements when traversing the list in the forward direction.
3) boolean hasPrevious(): Returns true if this list iterator has more elements when traversing the list in the reverse direction.
4) E next(): Returns the next element in the list and advances the cursor position.
5) int nextIndex(): Returns the index of the element that would be returned by a subsequent call to next().
6) E previous(): Returns the previous element in the list and moves the cursor position backwards.
7) int previousIndex(): Returns the index of the element that would be returned by a subsequent call to previous().
8) void remove(): Removes from the list the last element that was returned by next() or previous() (optional operation).
9) void set(E e): Replaces the last element returned by next() or previous() with the specified element (optional operation).