ListIterator

In the previous lesson, we looked at how to iterate an **ArrayList** using Iterator. The Iterator provides very limited capabilities as we can iterate only in the forward direction and we can't update or insert an element to the list while iterating. To overcome these problems, we can use ListIterator. The `listIterator()` method returns an object of type ListIterator which can then be used to iterate the **ArrayList**.

Below are the methods that are available in the ListIterator interface.

1. `hasNext()` - This method is used to check if there is a next element in the list when the list is iterated in the forward direction.
2. `next()` - This method returns the next element in the list and advances the cursor position.
3. `hasPrevious()` - This method is used to check if there is a next element in the list when the list is iterated in the backward direction.
4. `previous()` - This method returns the previous element in the list and moves the cursor position backward.
5. `nextIndex()` - This method returns the index of the element that would be returned by a subsequent call to `next()`. It returns the list size if the list iterator is at the end of the list.
6. `previousIndex()` - This method returns the index of the element that would be returned by a subsequent call to `previous()`. It returns -1 if the list iterator is at the beginning of the list.
7. `remove()` - This method removes the last element that was returned by `next()` or `previous()` from the list. This call can only be made once per call to `next()` or `previous()`. It can be made only if `add()` has not been called after the last call to `next()` or `previous()`.
8. `set(E e)` - This method replaces the last element returned by `next()` or `previous()` with the specified element. This call can be made only if neither `remove()` nor `add()` have been called after the last call to `next()` or `previous()`.
9. `add(E e)` - This method inserts the specified element into the list. The element is inserted immediately before the element that would be returned by `next()`, if any, and after the element that would be returned by `previous()`, if any.

The below example shows ListIterator working.

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

public class ArrayListDemo {

    public static void main(String args[]) {
        List<Integer> list = new ArrayList<>();
        list.add(10);
        list.add(20);
        list.add(30);
        list.add(40);

        // Getting ListIterator
        ListIterator<Integer> listIterator = list.listIterator();

        // Traversing elements
        System.out.println("Forward Direction Iteration:");
        while (listIterator.hasNext()) {
            System.out.println("Next element is " + listIterator.next() +
            " and next index is " + listIterator.nextIndex());

        }

        // Traversing elements, the iterator is at the end at this point
        System.out.println("Backward Direction Iteration:");
        while (listIterator.hasPrevious()) {
            System.out.println("Previous element is " + listIterator.previous() +
```

Run Save Reset

Why raw type Collection should be avoided

Whenever we create a Collection, we should provide the type of object it can hold. This is called **parameterized type** Collection. A **raw type** Collection does not have any type of safety, and an object of any type can be inserted into it. In the below example, we have created a **raw type** ArrayList. Elements of Integer and String type are added to it. This code will compile but will fail at run-time with `ClassCastException`. This would have been avoided if we had used **parameterized type**.

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

public class ArrayListDemo {

    public static void doSomeWork(List list) {
        list.add("India");
    }

    public static void main(String args[]) {
        List list = new ArrayList();
        list.add(10);
        list.add(20);
        doSomeWork(list);

        Integer i = (Integer) list.get(2);
    }
}
```

Run Save Reset