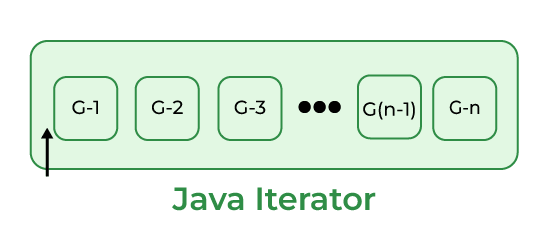
Different ways to access data present in the collection

1. normal loop
2. Using foreach loop

For index based accessing collections you can use for and for each loop

1. Using iterator ( front direction)

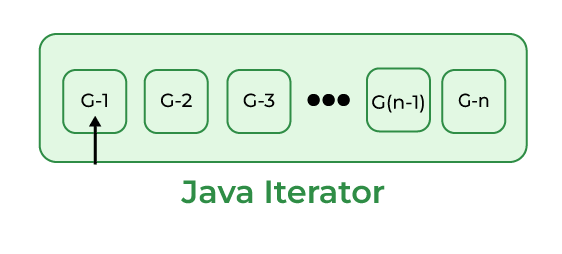
ArrayList al = new ArrayList();  
al.add(25);  
al.add(78);  
al.add(10);  
al.add(87);  
al.add(50);



Here Iterator’s Cursor is pointing before the first element of the List.

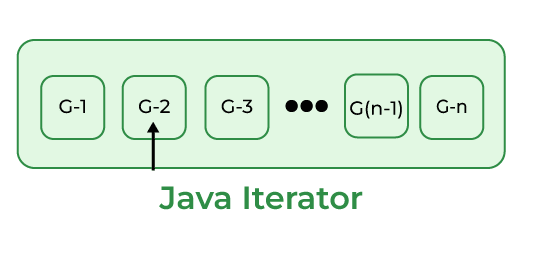
Now, we will run the following code snippet.

al.hasNext();  
al.next();

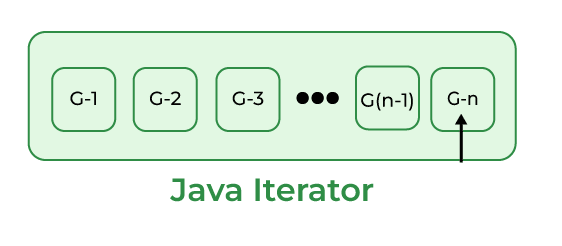


Now, we will run the following code snippet.

al.hasNext();  
al.next();

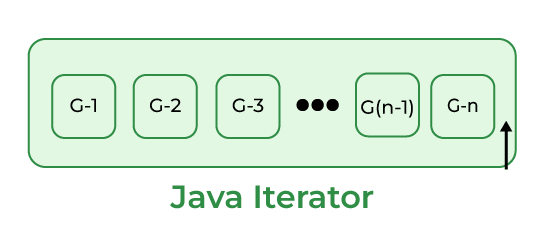


When we run the above code snippet, Iterator’s Cursor points to the second element in the list as shown in the above diagram. Do this process to reach the Iterator’s Cursor to the end element of the List.



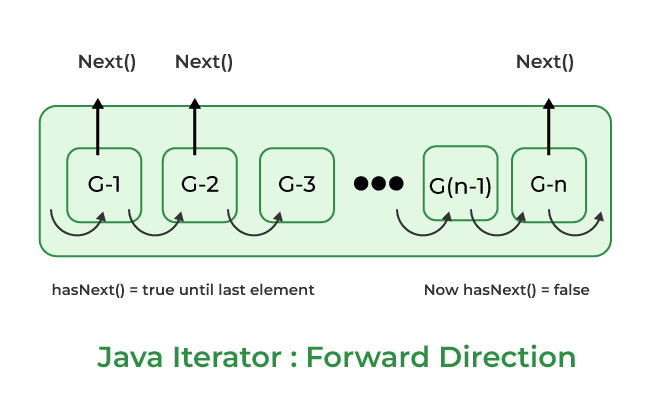
After reading the final element, if we run the below code snippet, it returns a “false” value.

al.hasNext();

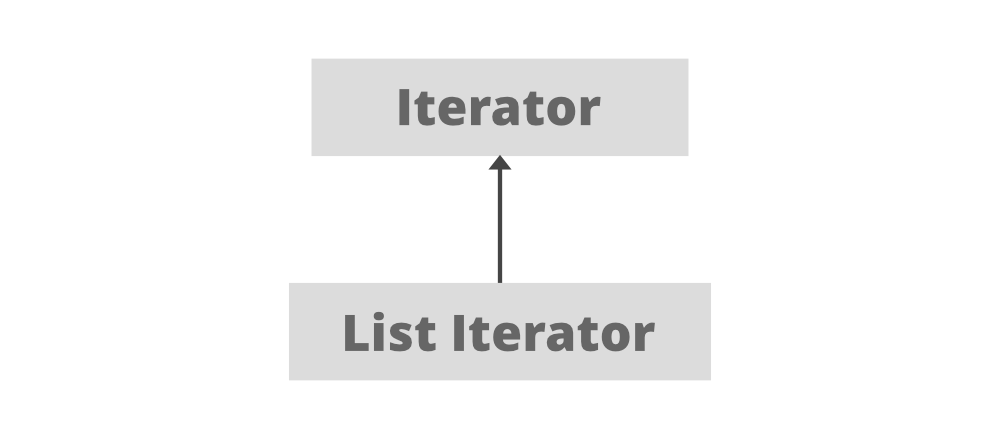


As Iterator’s Cursor points to the after the final element of the List, hasNext() method returns a false value.

Note: After observing all these diagrams, we can say that Java Iterator supports only Forward Direction Iteration as shown in the below diagram. So it is also known as Uni-Directional Cursor.

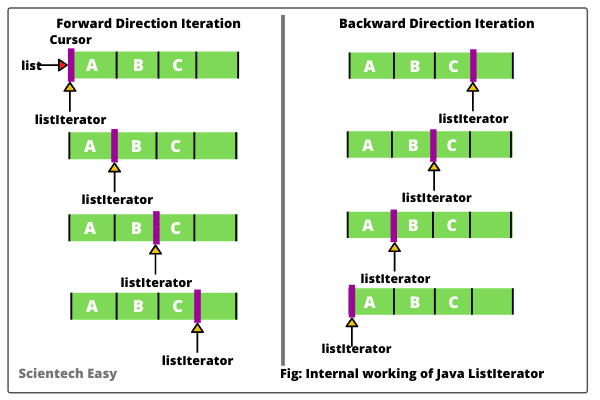


1. Using list iterator ( reverse direction )



Some Important points about ListIterator

1. It is useful for list implemented classes.
2. Available since java 1.2.
3. It supports bi-directional traversal. i.e both forward and backward directions.
4. It supports all the four CRUD operations(Create, Read, Update, Delete) operations.



**ListIterator is a bi-directional iterator. For this functionality, it has two kinds of methods:**

**1. Forward direction iteration**

* **hasNext():** This method returns true when the list has more elements to traverse while traversing in the forward direction
* **next():** This method returns the next element of the list and advances the position of the cursor.
* **nextIndex():** This method returns the index of the element that would be returned on calling the *next()* method.

**2. Backward direction iteration**

* **hasPrevious():** This method returns true when the list has more elements to traverse while traversing in the reverse direction
* **previous():** This method returns the previous element of the list and shifts the cursor one position backward.
* **previousIndex():** This method returns the index of the element that would be returned on calling the *previous()* method.

Eg: Ways\_For\_Traversing\_Collections

// go through the code