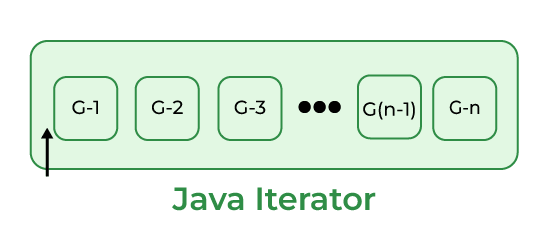
Different ways to access data present in the collection

1. normal loop

For index based accessing collections you can use for loop

1. Using foreach loop
2. Using iterator ( front direction)

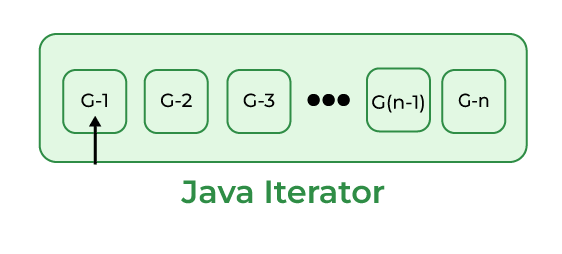
ArrayList al = new ArrayList();  
al.add(25);  
al.add(78);  
al.add(10);  
al.add(87);  
al.add(50);



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

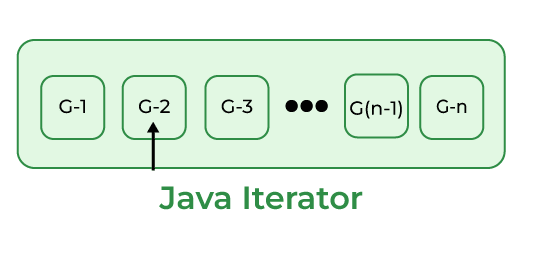
Now, we will run the following code snippet.

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

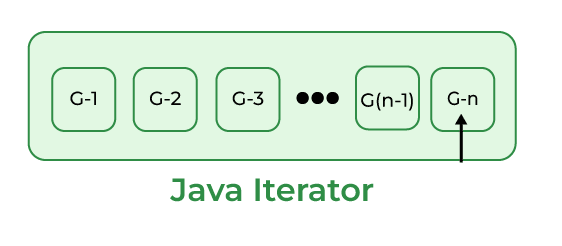


Now, we will run the following code snippet.

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

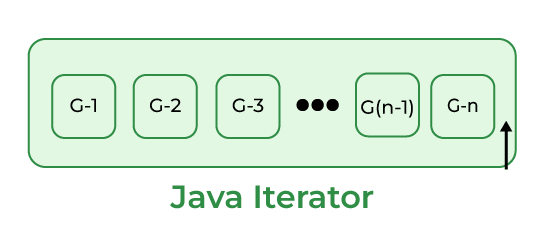


When we run the above code snippet, Iterator’s Cursor points to the second element in the list as shown in the above diagram. Do this process to reach the Iterator’s Cursor to the end element of the List.



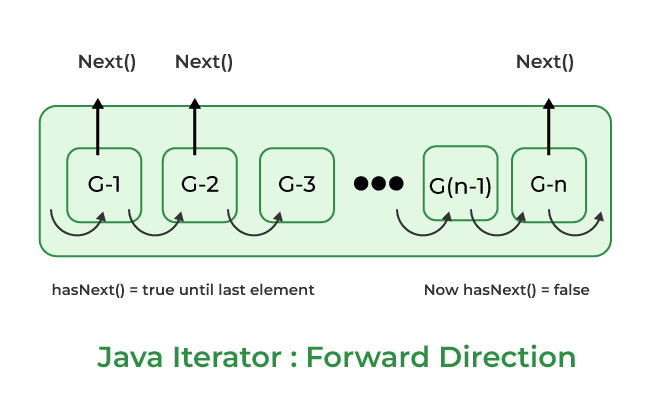
After reading the final element, if we run the below code snippet, it returns a “false” value.

al.hasNext();

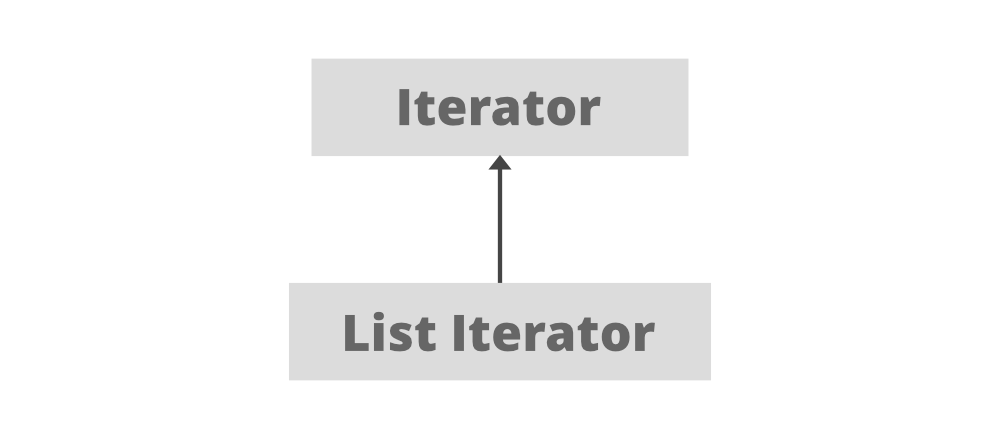


As Iterator’s Cursor points to the after the final element of the List, hasNext() method returns a false value.

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.

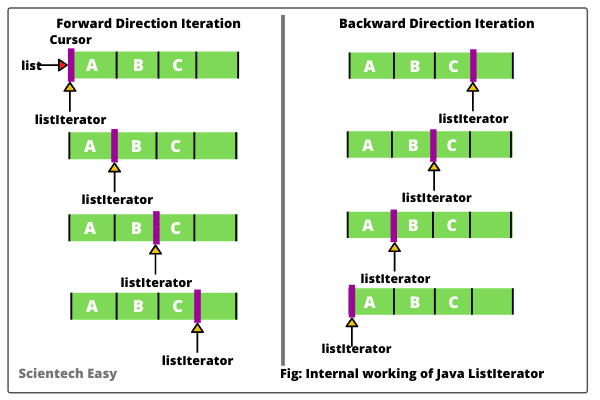


1. Using list iterator ( reverse direction )



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**

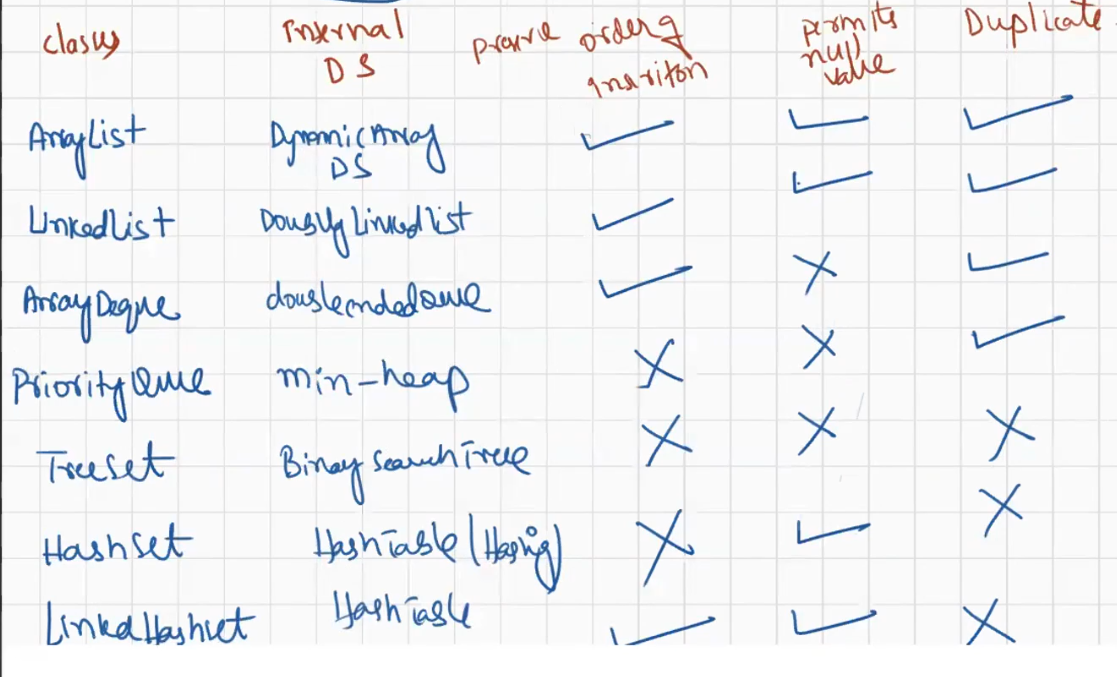
* **hasPrevious():** This method returns true when the list has more elements to traverse while traversing in the reverse direction
* **previous():** This method returns the previous element of the list and shifts the cursor one position backward.
* **previousIndex():** This method returns the index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

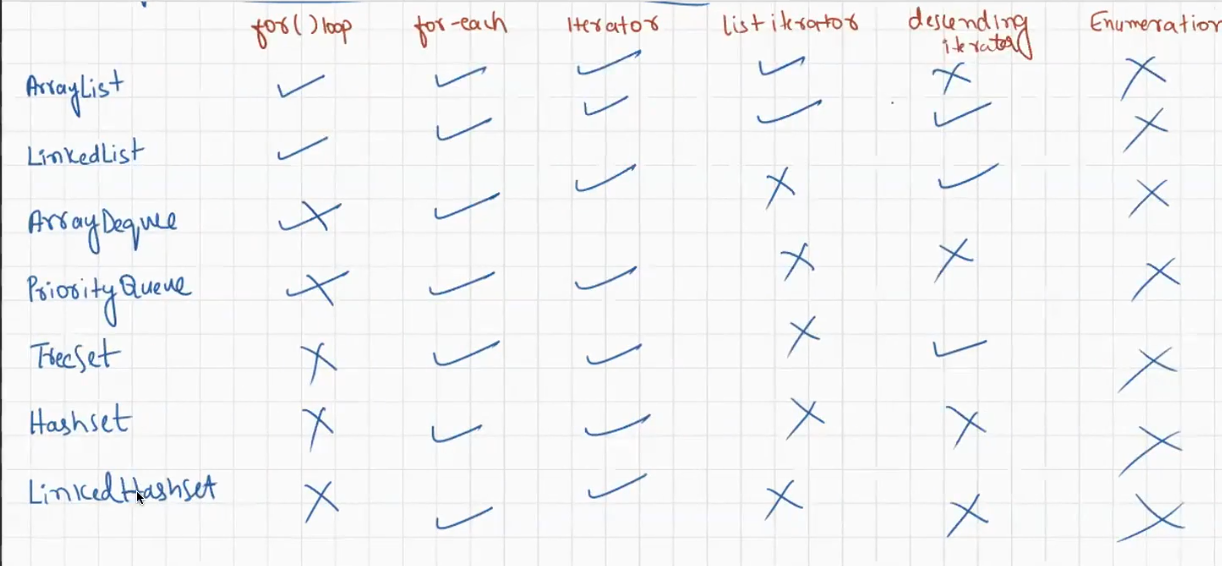
Eg: Ways\_For\_Traversing\_Collections

// go through the code

Eg: Vector\_Legacy\_Class

// go though the class





While you are trying the access the collection , if you attempt to modify the collection that is called structural modification or concurrent modification.

In for loop if you are attempting structural / concurrent modification it leads to infinite loop for this reason for loop is not suggestable if there is concurrent modification It is suggestable to use iterator.

Eg: Concurrent\_Modification\_Using\_For\_Loop

// go through the code

Fail Fast :

If you do the same with for each loop, then it will terminate the program with an exception which can be said as fail fast

Eg: Concurrent\_Modification\_Using\_For\_Each

If for each is used it will lead to ConcurrentModificationException.

If you do the same with iterator , then iterator will terminate the program with an exception

Reffered to fail fast

Eg: Concurrent\_Modification\_Using\_Iterator

Fail safe

Means if you try to make structural modification , that should not should happen and abnormal termination should also not happen for this in java a special package is present called concurrent package

Under concurrent package all the classes of collection are there.

Eg: Concurrent\_Modification\_With\_Concurrent\_Package

Collection: Collection is a [interface](https://www.geeksforgeeks.org/interfaces-in-java/) present in java.util package. It is used to represent a group of individual objects as a single unit. The collection is considered as the root interface of the collection framework. It provides several classes and interfaces to represent a group of individual objects as a single unit.

Collections: Collections is a utility class present in java.util package. It defines several utility methods like sorting and searching which is used to operate on collection. It has all static methods.

Collections methods will work only if objects are simple / homogenous data

If the objects are complex / heterogenous then comparable and comparator is used.

Eg: Collections\_Utility\_Class

If there is a mixed data in the collection like Integer , String , Long etc to sort Comparable and Comparator is used.

If you want to include only specific data type to the collection use generics

it is nothing but restricting the collection to a specific type.

Eg: Genrics\_Basic\_Example

//go through the code

Eg : Generics\_Basic\_Eg2

// go through the code