# Iterators in Java

Iterators are used in [Collection framework](https://www.geeksforgeeks.org/collections-in-java-2/) in Java to retrieve elements one by one. There are three iterators.

[**Enumeration**](https://www.geeksforgeeks.org/iterators-in-java/#Enumeration) **:**

It is a interface used to get elements of legacy collections(Vector, Hashtable). Enumeration is the first iterator present from JDK 1.0, rests are included in JDK 1.2 with more functionality. Enumerations are also used to specify the input streams to a *SequenceInputStream*. We can create Enumeration object by calling *elements()* method of vector class on any vector object

// Here "v" is an Vector class object. e is of// type Enumeration interface and refers to "v"Enumeration e = v.**elements**();

There are **two** methods in Enumeration interface namely :

// Tests if this enumeration contains more elements**public boolean hasMoreElements();**// Returns the next element of this enumeration // It throws **NoSuchElementException**// if no more element present**public Object nextElement();**

|  |
| --- |
| // Java program to demonstrate Enumeration  **import** java.util.Enumeration;  **import** java.util.Vector;    **public** **class** Test  {  **public** **static** **void** main(String[] args)  {  // Create a vector and print its contents  Vector v = **new** Vector();  **for** (**int** i = 0; i < 10; i++)  v.addElement(i);  System.out.println(v);    // At beginning e(cursor) will point to  // index just before the first element in v  Enumeration e = v.elements();    // Checking the next element availability  **while** (e.hasMoreElements())  {  // moving cursor to next element  **int** i = (Integer)e.nextElement();    System.out.print(i + " ");  }  }  } |

Output:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]0 1 2 3 4 5 6 7 8 9

**Limitations of Enumeration :**

* Enumeration is for **legacy** classes(Vector, Hashtable) only. Hence it is not a universal iterator.
* Remove operations can’t be performed using Enumeration.
* Only forward direction iterating is possible.

[**Iterator**](https://www.geeksforgeeks.org/iterators-in-java/#Iterator)**:**

It is a **universal** iterator as we can apply it to any Collection object. By using Iterator, we can perform both read and remove operations. It is improved version of Enumeration with additional functionality of remove-ability of a element.

Iterator must be used whenever we want to enumerate elements in all Collection framework implemented interfaces like Set, List, Queue, Deque and also in all implemented classes of Map interface. Iterator is the **only** cursor available for entire collection framework.

Iterator object can be created by calling *iterator()* method present in Collection interface.

// Here "c" is any Collection object. itr is of// type Iterator interface and refers to "c"Iterator itr = c.**iterator**();

Iterator interface defines **three** methods:

// Returns true if the iteration has more elements**public boolean hasNext();**// Returns the next element in the iteration// It throws **NoSuchElementException** if no more // element present**public Object next();**// Remove the next element in the iteration// This method can be called only once per call// to next()**public void remove();**

*remove()* method can throw two exceptions

* *UnsupportedOperationException :* If the remove operation is not supported by this iterator
* *IllegalStateException :* If the next method has not yet been called, or the remove method has already been called after the last call to the next method

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to demonstrate Iterator  **import** java.util.ArrayList;  **import** java.util.Iterator;    **public** **class** Test  {  **public** **static** **void** main(String[] args)  {  ArrayList al = **new** ArrayList();    **for** (**int** i = 0; i < 10; i++)  al.add(i);    System.out.println(al);    // at beginning itr(cursor) will point to  // index just before the first element in al  Iterator itr = al.iterator();    // checking the next element availabilty  **while** (itr.hasNext())  {  // moving cursor to next element  **int** i = (Integer)itr.next();    // getting even elements one by one  System.out.print(i + " ");    // Removing odd elements  **if** (i % 2 != 0)  itr.remove();  }  System.out.println();  System.out.println(al);  }  } |

Output:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]0 1 2 3 4 5 6 7 8 9 [0, 2, 4, 6, 8]

**Limitations of Iterator :**

* Only forward direction iterating is possible.
* Replacement and addition of new element is not supported by Iterator.

[**ListIterator**](https://www.geeksforgeeks.org/iterators-in-java/#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 List interface.

// Here "l" is any List object, ltr is of type// ListIterator interface and refers to "l"ListIterator ltr = l.**listIterator**();

ListIterator interface extends Iterator interface. So all three methods of Iterator interface are available for ListIterator. In addition there are **six** more methods.

// Forward direction// Returns true if the iteration has more elements**public boolean hasNext();**// same as s next() method of Iterator**public Object next();**// Returns the next element index // or list size if the list iterator// is at the end of the list**public int nextIndex();**// Backward direction// Returns true if the iteration has more elements// while traversing backward**public boolean hasPrevious();**// Returns the previous element in the iteration// and can throws **NoSuchElementException**// if no more element present**public Object previous();**// Returns the previous element index // or -1 if the list iterator is at the // beginning of the list**public int previousIndex();**// Other Methods // same as remove() method of Iterator**public void remove();**// Replaces the last element returned by // next() or previous() with the specified element **public void set(Object obj);**// Inserts the specified element into the list at// position before the element that would be returned // by next(),**public void add(Object obj);**

Clearly the three methods that *ListIterator* inherits from Iterator (*hasNext()*, *next()*, and *remove()*) do exactly the same thing in both interfaces. The *hasPrevious()* and the previous operations are exact analogues of *hasNext()* and *next()*. The former operations refer to the element before the (implicit) cursor, whereas the latter refer to the element after the cursor. The previous operation moves the cursor backward, whereas next moves it forward.

ListIterator has no current element; its cursor position always lies between the element that would be returned by a call to *previous()* and the element that would be returned by a call to *next()*

*set()* method can throw four exceptions

* *UnsupportedOperationException* – if the set operation is not supported by this list iterator
* *ClassCastException :* If the class of the specified element prevents it from being added to this list
* *IllegalArgumentException :* If some aspect of the specified element prevents it from being added to this list
* *IllegalStateException :* If neither next nor previous have been called, or remove or add have been called after the last call to next or previous

*add()* method can throw three exceptions

* *UnsupportedOperationException :* If the add method is not supported by this list iterator
* *ClassCastException :* If the class of the specified element prevents it from being added to this list
* *IllegalArgumentException :* If some aspect of this element prevents it from being added to this list

|  |
| --- |
| // Java program to demonstrate ListIterator  **import** java.util.ArrayList;  **import** java.util.ListIterator;    **public** **class** Test  {  **public** **static** **void** main(String[] args)  {  ArrayList al = **new** ArrayList();  **for** (**int** i = 0; i < 10; i++)  al.add(i);    System.out.println(al);    // at beginning ltr(cursor) will point to  // index just before the first element in al  ListIterator ltr = al.listIterator();    // checking the next element availabilty  **while** (ltr.hasNext())  {  // moving cursor to next element  **int** i = (Integer)ltr.next();    // getting even elements one by one  System.out.print(i + " ");    // Changing even numbers to odd and  // adding modified number again in  // iterator  **if** (i%2==0)  {  i++; // Change to odd  ltr.set(i); // set method to change value  ltr.add(i); // to add  }  }  System.out.println();  System.out.println(al);  }  } |

Output:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]0 1 2 3 4 5 6 7 8 9 [1, 1, 1, 3, 3, 3, 5, 5, 5, 7, 7, 7, 9, 9, 9]

**Limitations of ListIterator :** It is the most powerful iterator but it is only applicable for List implemented classes, so it is not a universal iterator.

**Important Common Points**

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

**2 :** We don’t create objects of Enumeration, Iterator, ListIterator because they are interfaces. We use methods like elements(), iterator(), listIterator() to create objects. These methods have anonymous [Inner classes](https://www.geeksforgeeks.org/inner-class-java/) that extends respective interfaces and return this class object. This can be verified by below code. For more on inner class refer

|  |
| --- |
| // Java program to demonstrate iterators references  **import** java.util.Enumeration;  **import** java.util.Iterator;  **import** java.util.ListIterator;  **import** java.util.Vector;    **public** **class** Test  {  **public** **static** **void** main(String[] args)  {  Vector v = **new** Vector();    // Create three iterators  Enumeration e = v.elements();  Iterator itr = v.iterator();  ListIterator ltr = v.listIterator();    // Print class names of iterators  System.out.println(e.getClass().getName());  System.out.println(itr.getClass().getName());  System.out.println(ltr.getClass().getName());  }  } |

Output:

java.util.Vector$1java.util.Vector$Itrjava.util.Vector$ListItr

The **$** symbol in reference class name is a proof that concept of inner classes is used and these class objects are created.