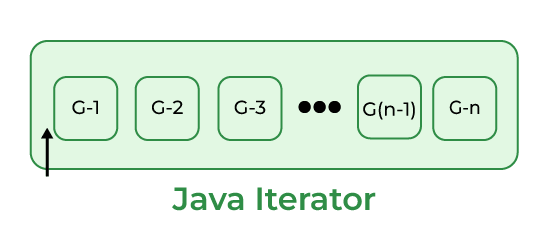
Different ways to access data present in the collection

1. normal loop

For index based accessing collections you can use for loop

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

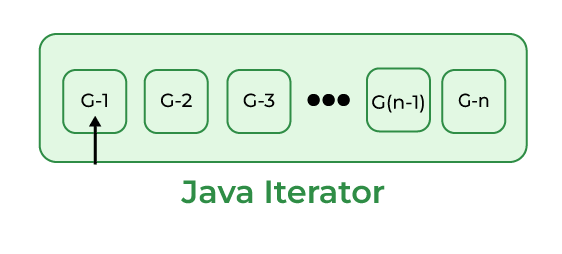
ArrayList al = new ArrayList();  
al.add(25);  
al.add(78);  
al.add(10);  
al.add(87);  
al.add(50);



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

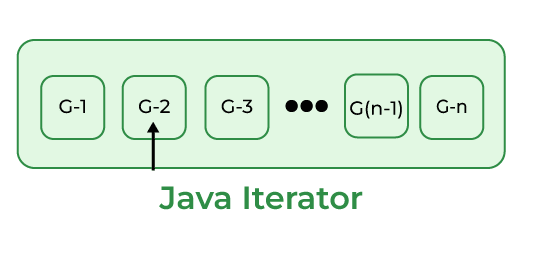
Now, we will run the following code snippet.

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

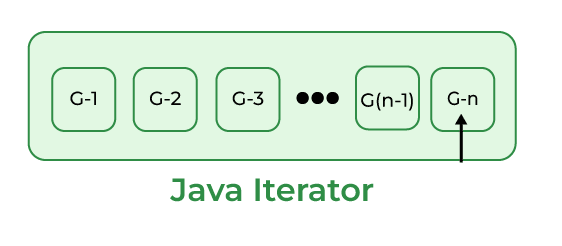


Now, we will run the following code snippet.

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

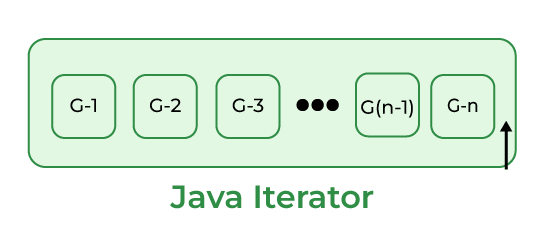


When we run the above code snippet, Iterator’s Cursor points to the second element in the list as shown in the above diagram. Do this process to reach the Iterator’s Cursor to the end element of the List.



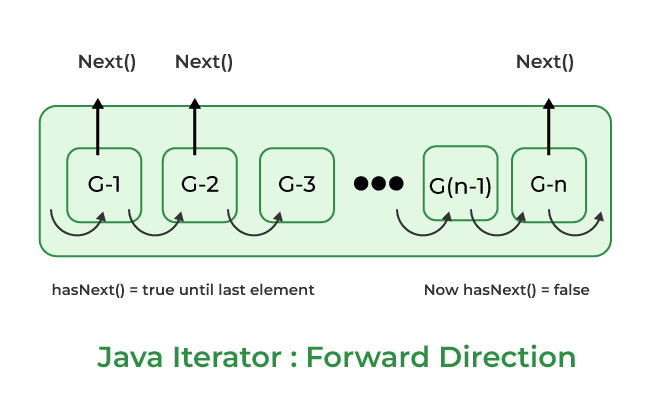
After reading the final element, if we run the below code snippet, it returns a “false” value.

al.hasNext();

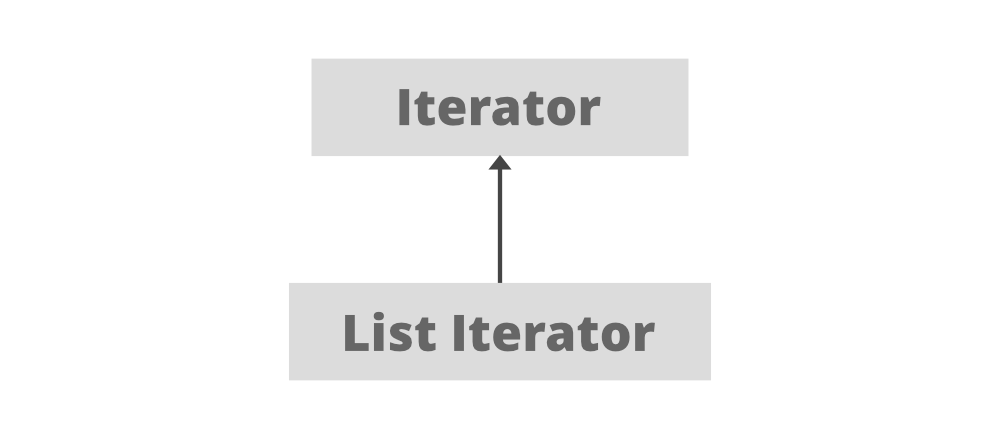


As Iterator’s Cursor points to the after the final element of the List, hasNext() method returns a false value.

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.

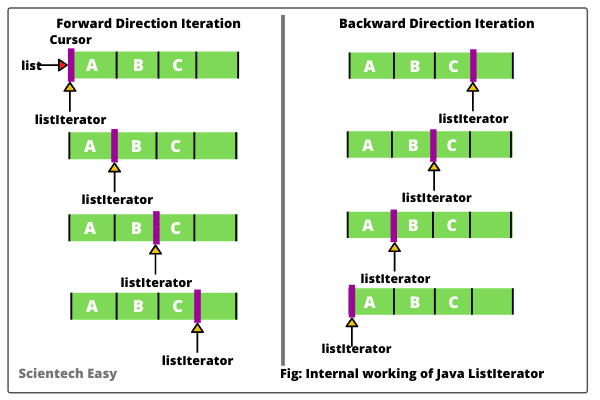


1. Using list iterator ( reverse direction )



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**

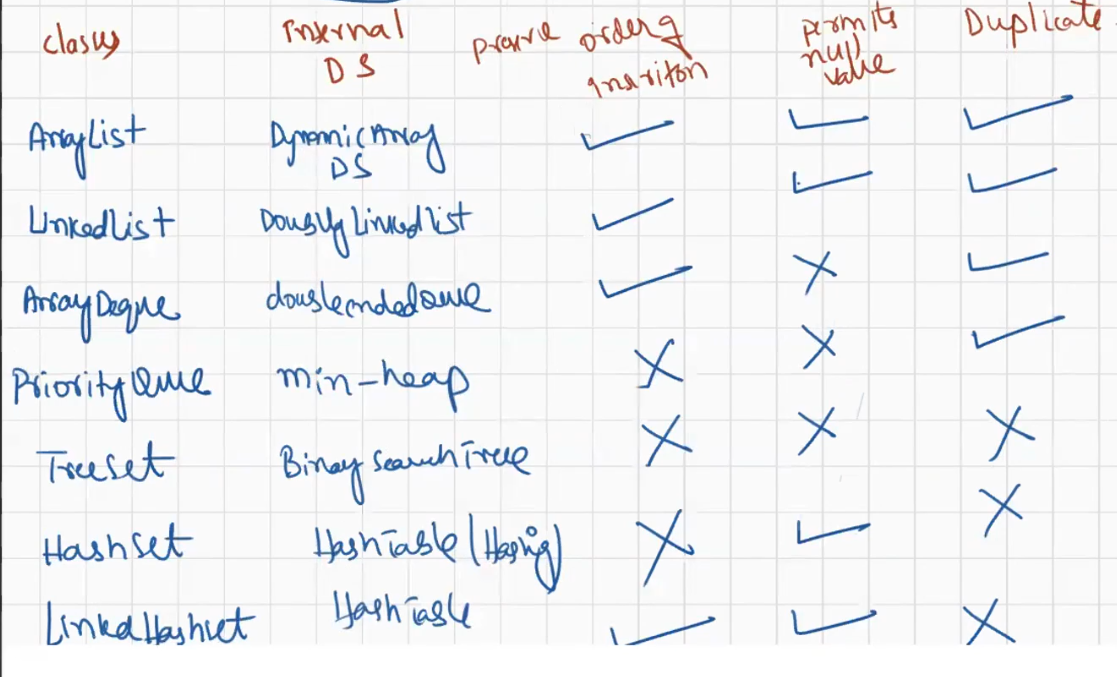
* **hasPrevious():** This method returns true when the list has more elements to traverse while traversing in the reverse direction
* **previous():** This method returns the previous element of the list and shifts the cursor one position backward.
* **previousIndex():** This method returns the index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

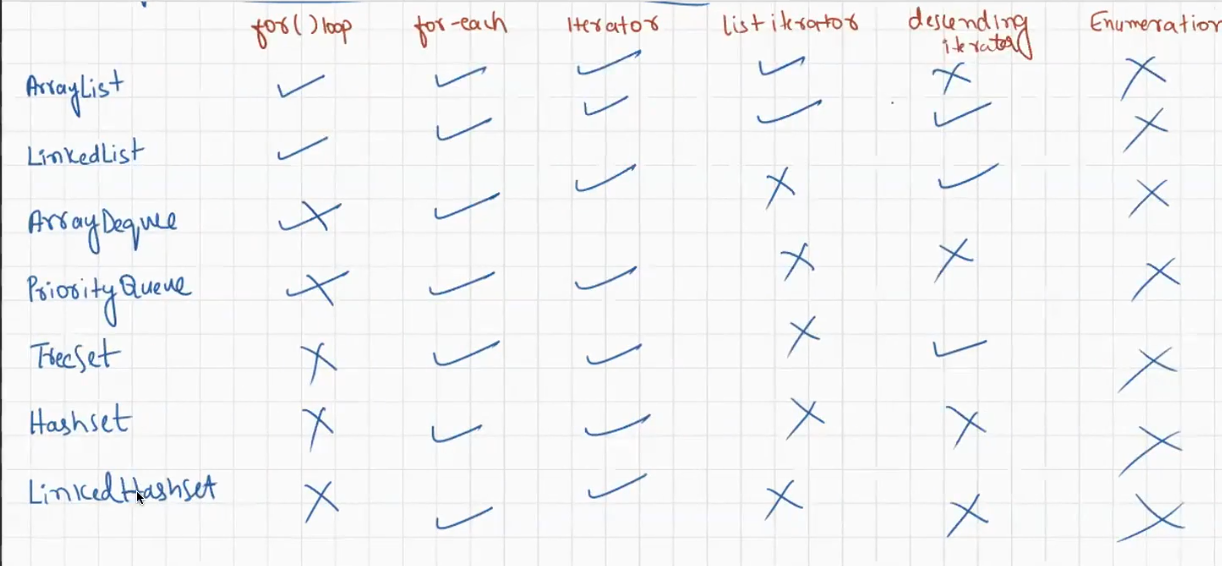
Eg: Ways\_For\_Traversing\_Collections

// 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

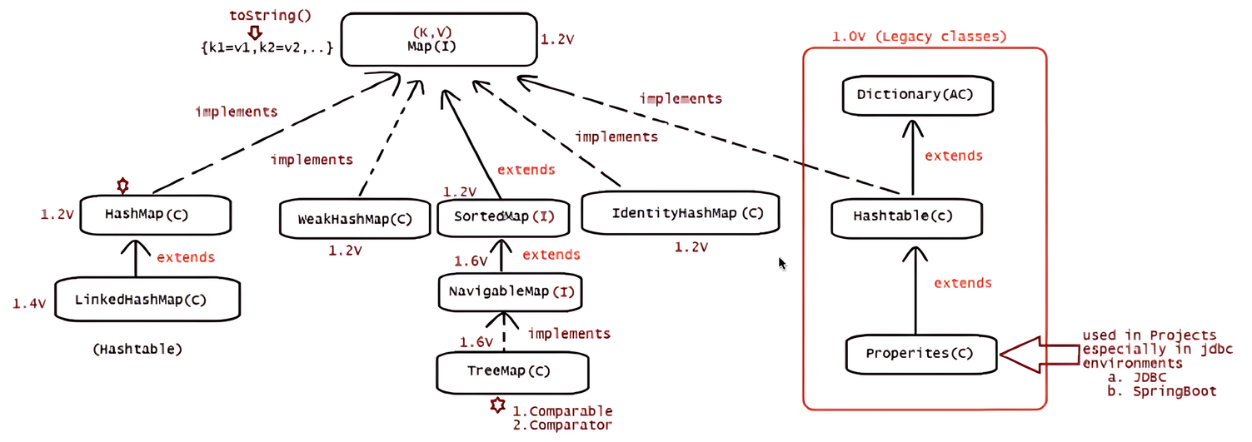
Eg : Generics\_Basic\_Eg2

// go through the code

Hashing load factor 0.75 or 75 , if 75 percent of buckets are filled the size will be doubled.

Searching operation will be very fast .

In Java, Map Interface is present in [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package represents a mapping between a key and a value. Java Map interface is not a subtype of the [Collection interface](https://www.geeksforgeeks.org/collections-in-java-2/).



why and when to use Maps

The maps are used to perform lookups by keys or when someone wants to retrieve and update elements by keys. Some common scenarios are as follows:

* A map of error codes and their descriptions.
* A map of zip codes and cities.
* A map of managers and employees. Each manager (key) is associated with a list of employees (value) he manages.
* A map of classes and students. Each class (key) is associated with a list of students (value).
* The key has to be unique and it can be of any data type
* The value need not to be unique and it can be of any data type

HashMap:

Order of insertion is not maintained

Java HashMap class implements the Map interface which allows us to store key and value pair. Each key and value pair is called as an entry.

HashMaps allow for duplicate values, but not duplicate keys. If a duplicate key is added, the previous value associated with the key is overwritten.

Internally uses hashes tables data structure

HashMaps retrieval and insertion of elements are very fast, usually O(1) time complexity.

It allows to store the null keys as well, but there should be only one null key object and there can be any number of null values.

Interface Map{

Interface Entry{

}

}

Each key and value pair is called as an entry. Entry is a nested Interface of Map. Entry is tightly coupled with Map.

Key and value is treated as object in HashMap.

Eg: HashMap\_Eg1

// go through the code

Eg: Acessing\_Elements\_In\_HashMap\_With\_InBuilt\_Methods

// go through the code

Note:

System.out.println() internally calls tostring() so irrespective of the data you are seeing on the screen , it is string only.

Some classes will not override tostring() method so we get address / hashcode as output

In some classes toString method is overridden so we will get the values

Note: Generally in the industry System.out.println() method is used for debugging purpose. Means whether the control comes to particular area . and what type a method is returning

Eg: Overriding\_ToString\_Method\_In\_HashMap

// go through the code

Hashtable:

* It is similar to HashMap, but is synchronized.

The Hashtable class implements a hash table, which maps keys to values.

Any non-null object can be used as a key or as a value.

However, the use of `Hashtable` class has been discouraged generally. This is primarily due to its synchronized nature by default, which can result in slower performance compared to other implementations of the `Map` interface like `HashMap`

In general, it’s recommended to use the Map interface or one of its implementations (such as HashMap or ConcurrentHashMap) instead of the Hashtable class.

LinkedHashMap :

The LinkedHashMap Class is just like [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/) with an additional feature of maintaining an order of elements inserted into it. HashMap provided the advantage of quick insertion, search, and deletion but it never maintained the track and order of insertion, which the LinkedHashMap provides where the elements can be accessed in their insertion order.

Features of a LinkedHashMap:

* A LinkedHashMap contains values based on the key. It implements the Map interface and extends the HashMap class.
* It contains only unique elements.
* It may have one null key and multiple null values.
* It is non-synchronized.

Eg: LinkedHashMap\_Eg1

// go through the code

Eg: Acessing\_Elements\_In\_LinkedHashMap\_With\_InBuilt\_Methods

// go through the code

Eg: Overriding\_ToString\_Method\_In\_LinkedHashMap

// go through the code

 Hasmap does not support iterator() , values() is used it retuens collection . and on collection we can use iteratot concept

Print stmnt prints the data of hashmap , but if you want to use the data some where in the program accessing is needed.

Values(), keyset() , entryset().

Eg: HashMap\_GetKey\_GetValue\_Methods

// go through the code

Garbage collector

Internally garbage collector calls finalize() method .

Eg: Garbage\_Collector\_Finalize\_Method

// go through the code

Note: Garbage collector logic is present in the finally method.

Eg: HashMap\_Dominating\_Garbage\_Collector

// go through the code

Even though the object is specified as key in hashmap, it does not have any reference and it is not eligible for garbage collection if it is associated with HashMap i.e. HashMap dominates over Garbage Collector.

WeakHashMap :

 WeakHashMap is almost same as HashMap except in case of WeakHashMap, if object is specified as key doesn’t contain any references- it is eligible for garbage collection even though it is associated with WeakHashMap.

A few important features of a WeakHashMap Class are:

* Both null values and null keys are supported in WeakHashMap.
* It is not synchronized.
* Eg: Garbage\_Collector\_Dominating\_WeakHashMap