Accessing a Java Collection using Iterators

To access elements of a collection, either we can use index if collection is list based or we need to traverse the element. There are three possible ways to traverse through the elements of any collection.

1. Using Iterator interface
2. Using ListIterator interface
3. Using for-each loop

Accessing elements using Iterator

Iterator is an interface that is used to iterate the collection elements. It is part of java collection framework. It provides some methods that are used to check and access elements of a collection.

Iterator Interface is used to traverse a list in forward direction, enabling you to remove or modify the elements of the collection. Each collection classes provide `iterator()` method to return an iterator.

Iterator Interface Methods

Method	Description
boolean `hasNext()`	Returns **true** if there are more elements in the collection. Otherwise, returns false.
E `next()`	Returns the **next element present** in the collection. Throws NoSuchElementException if there is not a next element.
void `remove()`	Removes the current element. Throws IllegalStateException if an attempt is made to call remove() method that is not preceded by a call to `next()` method.

Iterator Example

In this example, we are using iterator() method of collection interface that returns an instance of Iterator interface. After that we are using hasNext() method that returns true of collection contains an elements and within the loop, obtain each element by calling `next()` method.

```
import java.util.*;

class Demo

{

  public static void main(String[] args)
```

```
{
  ArrayList< String> ar = new ArrayList< String>();
  ar.add("ab");
  ar.add("bc");
  ar.add("cd");
  ar.add("de");
  Iterator it = ar.iterator();    //Declaring Iterator
  while(it.hasNext())
  {
    System.out.print(it.next()+" ");
  }
 }
}
```

Copy

```
ab bc cd de
```

Accessing elements using `ListIterator`

`ListIterator` Interface is used to traverse a list in both **forward** and **backward** direction. It is available to only those collections that implements the **List** Interface.

Methods of ListIterator:

Method	Description
void `add(E obj)`	Inserts obj into the list in front of the element that will be returned by the next call to next() method.
boolean `hasNext()`	Returns true if there is a next element. Otherwise, returns false.
boolean `hasPrevious()`	Returns true if there is a previous element. Otherwise, returns false.

E next()	Returns the next element. A NoSuchElementException is thrown if there is not a next element.
int nextIndex()	Returns the index of the next element. If there is not a next element, returns the size of the list.
E previous()	Returns the previous element. A NoSuchElementException is thrown if there is not a previous element.
int previousIndex()	Returns the index of the previous element. If there is not a previous element, returns -1.
void remove()	Removes the current element from the list. An IllegalStateException is thrown if remove() method is called before next() or previous() method is invoked.
void set(E obj)	Assigns obj to the current element. This is the element last returned by a call to either next() or previous() method.

ListIterator Example

Lets create an example to traverse the elements of ArrayList. ListIterator works only with list collection.

```
import java.util.*;

class Demo

{

  public static void main(String[] args)

  {

    ArrayList< String> ar = new ArrayList< String>();

    ar.add("ab");

    ar.add("bc");

    ar.add("cd");

    ar.add("de");
```

```
    ListIterator litr = ar.listIterator();

    while(litr.hasNext())    //In forward direction

    {

      System.out.print(litr.next()+" ");

    }

    while(litr.hasPrevious())    //In backward direction

    {

      System.out.print(litr.previous()+" ");

    }

  }

}
```

Copy

```
ab bc cd de de cd bc ab
```

for-each loop

for-each version of for loop can also be used for traversing the elements of a collection. But this can only be used if we don't want to modify the contents of a collection and we don't want any **reverse** access. for-each loop can cycle through any collection of object that implements Iterable interface.

Exmaple:

```
import java.util.*;

class Demo

{

  public static void main(String[] args)

  {

    LinkedList< String> ls = new LinkedList< String>();

    ls.add("a");

    ls.add("b");

    ls.add("c");

    ls.add("d");

    for(String str : ls)

    {
```

```
        System.out.print(str+" ");

      }

   }

}
```

Copy

```
a b c d
```

Traversing using for loop

we can use for loop to traverse the collection elements but only index-based collection can be accessed. For example, list is index-based collection that allows to access its elements using the index value.

```
import java.util.*;

class Demo

{

  public static void main(String[] args)

  {

    LinkedList<String> ls = new LinkedList<String>();

    ls.add("a");

    ls.add("b");

    ls.add("c");

    ls.add("d");

    for(int i = 0; i<ls.size(); i++)

    {

      System.out.print(ls.get(i)+" ");

    }

   }

}
```

Copy

```
Output-

a b c d
```