Java – How to use Iterator?

Often, you will want to cycle through the elements in a collection. For example, you might want to display each element.

The easiest way to do this is to employ an iterator, which is an object that implements either the Iterator or the ListIterator interface.

"An iterator is an object that allows a programmer to traverse through all the elements of a collection, regardless of its specific implmentation.[1]" This allows the user to traverse through a collection of objects, without them having to know the details of the implementation (i.e. they don't know if data are stored in an array, ArrayList, linked list, etc.)

Iterator enables you to cycle through a collection, obtaining or removing elements. ListIterator extends Iterator to allow bidirectional traversal of a list, and the modification of elements.

Before you can access a collection through an iterator, you must obtain one. Each of the collection classes provides an iterator( ) method that returns an iterator to the start of the collection. By using this iterator object, you can access each element in the collection, one element at a time.

In general, to use an iterator to cycle through the contents of a collection, follow these steps:

> Obtain an iterator to the start of the collection by calling the collection's iterator( ) method.
>
> Set up a loop that makes a call to hasNext( ). Have the loop iterate as long as hasNext( ) returns true.
>
> Within the loop, obtain each element by calling next( ).

For collections that implement List, you can also obtain an iterator by calling ListIterator.

The Methods Declared by Iterator:

SN	Methods with Description
1	**boolean hasNext( )** Returns true if there are more elements. Otherwise, returns false.
2	**Object next( )** Returns the next element. Throws NoSuchElementException if there is not a next element.
3	**void remove( )** Removes the current element. Throws IllegalStateException if an attempt is made to call remove( ) that is not preceded by a call to next( ).

[1] From Wikipedia as attributed on http://blog.dreasgrech.com/2010/03/javas-iterators-and-iterables.html

http://www.tutorialspoint.com/java/java_using_iterator.htm
http://tutorials.jenkov.com/java-collections/iterable.html

Example: of use on an ArrayList (all ADTs in the Collections class can return an Iterator, which can be used similarly).

```
import java.util.*;

public class IteratorDemo {

   public static void main(String args[]) {
      // Create an array list
      ArrayList al = new ArrayList();
      // add elements to the array list
      al.add("C");
      al.add("A");
      al.add("E");
      al.add("B");
      al.add("D");
      al.add("F");

      // Use iterator to display contents of al
      System.out.print("Original contents of al: ");
      Iterator itr = al.iterator();
      while(itr.hasNext()) {
         Object element = itr.next();
         System.out.print(element + " ");
      }
      System.out.println();
```

…

ListIterator provides additional methods that includes hasPrevious, nextIndex, previous, set, add, etc.

What's the Iterable interface for?

The Iterable interface (java.lang.Iterable) is one of the root interfaces of the Java collection classes. The Collection interface extends Iterable, so all subtypes of Collection also implement the Iterable interface.

A class that implements the Iterable can be used with the enhanced for-loop. Here is such an example:

```
List list = new ArrayList();

for(Object o : list){
    //do something o;
}

The Iterable interface has only one method:

public interface Iterable<T> {
  public Iterator<T> iterator();
}
```

http://www.tutorialspoint.com/java/java_using_iterator.htm
http://tutorials.jenkov.com/java-collections/iterable.html

The Iterable interface allows programmers to use the enhanced for-loop with their own custom classes. This is accomplished by implementing the **`java.lang.Iterable<E>`** interface.

Example: (from http://tutorials.jenkov.com/java-generics/implementing-iterable.html)

```
public class MyCollection<E> implements Iterable<E>{

    public Iterator<E> iterator() {
        return new MyIterator<E>();
    }
}
```

--
And here is the corresponding implementation skeleton of the MyIterator class:

```
public class MyIterator <T> implements Iterator<T> {

    public boolean hasNext() {

        //implement...
    }

    public T next() {
        //implement...;
    }

    public void remove() {
        //implement... if supported.
    }
}
```

The specific implementations of hasNext and next depends on how the data in MyCollection is stored.

Here is how to use the MyCollection generified, with the enhanced for-loop:

```
public static void main(String[] args) {
    MyCollection<String> stringCollection = new MyCollection<String>();

    for(String string : stringCollection){
      // do something with each "string"
    }
}
```

The compiler changes the code in the "for" immediately above to:

```
Iterator<String> itr = stringCollection.iterator( );
while (itr.hasNext( )) {
      String string = itr.next( );
      // do something with "string"
}
```

In the Weiss text, Professor Weiss illustrates the use of Iterator and Iterable in the implementation of `MyArrayList` on pages 69 & 70.

http://www.tutorialspoint.com/java/java_using_iterator.htm
http://tutorials.jenkov.com/java-collections/iterable.html