Assignment 6

Rasika Mohod
rmohod@gmu.edu
G01044774

Iterator and Iterable Interfaces: Usage and Implementation

The main aim of this article is to define and discuss properties of two different interfaces in java; java.util.Iterator and java.lang.Iterable. Introduced in the Java JDK 1.2 release, the java.util.Iterator interface allows the iteration of container classes. And Iterable, introduced in J2SE 5.0 release of Java, is an interface to support an enhanced loop for iterating over collection and arrays. This paper clearly defines Iterator and Iterable interfaces and states their origin and evolution in java language. It also discusses the usages of these two interfaces in different scenarios along with notable differences in their implementation by providing specific examples.

Introduction:

Iterators are the generalization of the iteration mechanism which is found to be available in most programming languages along with java. They allow the users to iterate over arbitrary types of data in a convenient and efficient way. An iterator is an Object, which enables a Collection<E> to be traversed. It allows developers to retrieve data elements in a Collection without any knowledge of the underlying data structure, whether it is an ArrayList, LinkedList, Set or some other kind. The point of using an iterator is to allow the client to iterate over the collection of objects, without exposing implementation details. This gives user the benefit to change what collection the Iterator iterates over and how each element is served, without making any change in the client's code.

The **Iterable** interface was introduced to support an enhanced for (foreach) loop for iterating over collections and arrays. An Iterable is a simple representation of a series of elements that can be iterated over. It does not have any iteration state such as a "current element". It is a container class and has one method named iterator() that produces an Iterator. Hence, any class that implements Iterable will return an Iterator. Iterable can produce any number of valid Iterators.

API History:

The API *java.util.Iterator* and *java.lang.Iterable* are introduced and used in Java for a while now. The Iterator interface came into existence earlier than the Iterable interface. It is stated that the origination of Iterator was in 1.2 Java release and the class called Iterable was introduced in Java 1.5. The public interface Iterator<E>, an iterator over a collection took place of Enumeration in the Java Collections Framework. The two new methods of Iterable interface, forEach(Consumer<? super T> action) and spliterator() were introduced in Java 1.8.

Usage and Implementation:

Interface Iterator<E>:

The concept behind iterator is very simple: An Iterator is always returned by a Collection and has methods such as next() which returns the next element in the Collection, hasNext() which returns a boolean value indicating if there are more elements in the Collection or not, etc. Iterators promote loose coupling between Collection classes and the classes using these Collections. For example, a class containing some kind of an algorithm (e.g. Search) is only concerned with traversing the list without knowing the exact list structure or any low level details.

All Iterator objects have to implement this interface and are bound by its protocols. The interface has four methods:

```
public interface Iterator<E> {
      boolean hasNext();
      E next();
      void remove();
      void forEachRemaining(Consumer<? super E> action);
}
```

- next() : Returns the next element in the Collection. To get all elements in a Collection, call this method repeatedly. When the end is reached and there are no more elements present, this method throws *NoSuchElementException*.
- hasNext() : Returns true if the Collection has more elements, returns false otherwise.
- remove() : Removes the last returned element from the Collection.
- forEachRemaining(Consumer<? super E> action) : Performs the given action for each remaining element until all elements have been processed or the action throws an exception.

A java.util.Iterator can only move in one direction: forward. Once the iterator has reached the end of the list, it cannot be reset to the starting position again. In this case, a new Iterator should be obtained.

Consider following **example** of an iterator traversing over a Collection of ArrayList to print its elements:

```
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("A");
      al.add("B");
      al.add("C");
      al.add("D");
      al.add("E");
      al.add("F");

      // Use iterator to display contents of array
      System.out.print("Original contents of array: ");
      Iterator itr = al.iterator();

      while(itr.hasNext()) {
         Object element = itr.next();
         System.out.print(element + " ");
      }
   }
}
```

<u>Interface Iterable<T>:</u>

It is a very simple Interface which defines three methods:

```
public interface Iterable<T> {
      Iterator<T> iterator();
      void forEach(Consumer<? super T> action);
      Spliterator<T> spliterator();
}
```

- iterator() : Returns an iterator over elements of type T.
- forEach(Consumer<? super T> action) : Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception. Unless otherwise specified by the implementing class, actions are performed in the order of iteration (if an iteration order is specified). Exceptions thrown by the action are relayed to the caller.
- spliterator() : Creates a Spliterator over the elements described by this Iterable.

The main purpose of this interface is to allow Objects implementing it to be used in for-each loop. Also, Iterable can be rewound. A programmer can rewind an Iterable by getting a new iterator() from the Iterable. However as an Iterator only has the next() method, thus an Iterable is handy if the need is to traverse the elements more than once.

Consider the flowing **example** of Iterable interface for iterating over a List collection with String object elements:

```
public class MyIterable<T> implements Iterable<T> {

      private List<T> list;

      public MyIterable(T [] t) {
            list = Arrays.asList(t);
            Collections.reverse(list);
      }

      @Override
      public Iterator<T> iterator() {
            return list.iterator();
      }

      public static void main(String [] args) {
            //Create Integers List
            Integer [] ints = {1, 2, 3};
            MyIterable<Integer> myList = new MyIterable<>(ints);

            //Traverse the list and print the elments of the list
            for (Integer i : myList) {
                  System.out.println(i);
            }
      }
}
```

The **difference** between the Iterator and Iterable can also be inferred by considering the example of an interface verses a concrete class. From the API documentation, it has been noticed that usually objects that implements some interface ensures that the certain collections of the methods will be available. The API documentation of both of the categories depict that Iterator<E> represents the type of elements returned by *this* iterator whereas Iterable<T> represents the type of elements returned by *the iterator*.

Conclusion:

Iterable and Iterator interfaces in java work well hand-in-hand and are used together extensively to reduce many computations in the need of iterating over a collection again and again. While there exists the difference between these two interfaces, it is very important to understand their definition, description, differences, usage and implementation for concise and proper use. At last, user should be clear and thoroughly aware of the fact that Iterator interface does not extend Iterable and could simply return 'this' while iterator can be called multiple times returning a fresh iterator each time.

References:

Book: Program Development in Java: Abstraction, Specification, and Object Oriented Design - Barbara Liskov; John Guttag.

https://10kloc.wordpress.com/2012/12/15/iterators-in-java-listiterator-iterable-as-well/

https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html

https://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html

https://en.wikipedia.org/wiki/Iterator

https://www.tutorialspoint.com/