Lecture 8: Iterating Over Collections

Reading materials    Dale, Joyce, Weems:
OpenDSA: Not covered.
Liang: in Comprehensive edition only

Topics Covered

1 Introduction 1

2 Iterable<T> Interface 1

3 Iterator<T> interface 2

4 For - Each Loop in Java 2

5 Implementing Iterators 3
   5.1 Internal Iterators 3

1 Introduction

When using a collection data structure (any data structure that contains a, possibly large, number of data items organized in some fashion), it would be nice to have a uniform way of iterating over, or accessing the data items without worry as to how they are stored and in what order.

An iterator allows the client of a collection data structure to ”walk” through it in some predetermined (by the data structure itself) order without exposing the details of the data structure itself. All collection classes in Java implement the Iterable interface.

2 Iterable<T> Interface

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

Any class that implements the generic Iterable<T> interface has to implement a method called iterator():

    Iterator<T> iterator() Returns an iterator over a set of elements of type T.

The type T stands for whatever type of data is stored in the collection. T can be replaced by any type without restriction.

Implementing Iterable<T> interface allows an object (a data structure object) to be the target of the for each loop statement.
3 Iteration<T> interface

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

Collections that implement Iterable<T> interface need to provide a method that returns an object that implements the Iterator<T> interface. The objects that implement the Iterator<T> interface are commonly just called iterators. They provide a way of traversing a collection without knowing its underlying structure. The iterators do not store the data on their own. Instead, they maintain references to places in the collection at which the actual data is stored.

The Iterator<T> interface requires that the following methods are implemented by the classes that implement the interface:

- hasNext() returns true, if there are any more elements in the collection to iterate over.
- next() returns a reference to the next element in the collection (the order of element traversal is determined internally by the collection itself). If there is no next element, the method throws NoSuchElementException exception.
  
  WARNING: any call to next() should always be preceded by a call to hasNext(). For example one could iterate over an underlying collection using an iterator iter as follows:

  ```java
  while ( iter.hasNext() ) {
    data = iter.next();
    // do something to data
  }
  ```

- remove() removes the last element returned by next() from the underlying collection.
  
  WARNING: remove() can be called only once per each call to next(). For example one could iterate over an underlying collection using an iterator iter and remove every single item from the collection using:

  ```java
  while ( iter.hasNext() ) {
    data = iter.next();
    iter.remove();
  }
  ```

  The IllegalStateException exception is thrown if remove() is called more than once per call to next() or before any call to next() is made.

  OPTIONAL: the remove() operation of the iterator is considered to be optional. If it is not implemented, calling remove() does not change the underlying collection, but it should throw UnsupportedOperationException.

4 For - Each Loop in Java

http://docs.oracle.com/javase/8/docs/technotes/guides/language/foreach.html

The for-each loop in Java uses the iterators of collection classes to iterate over all elements of the collection. We never create an iterator, but it is created and used internally when running one of those loops.

Syntax:

```java
for ( Type item : CollectionName ) {
```
//do something with each item
}

For an example of use, see http://docs.oracle.com/javase/1.5.0/docs/guide/language/foreach.html.

5 Implementing Iterators

There are three different approaches of implementing iterators for our own collection classes:

- internal iterators - using inner classes,
- external iterators - using external classes,
- turning collections into their own iterators.

5.1 Internal Iterators

An internal iterator is implemented as an inner or nested class of the contained class.

See http://docs.oracle.com/javase/tutorial/java/javaOO/nested.html for a brief tutorial on nested classes in Java.

This gives the iterator class access to all data fields and methods of the contained class (the details of which should not be shared with the "outside world").

See TestList.java, List.java and MyInteger.java for a simple example of use of an internal iterator.