Interface Iterator

The words *enumerate* and *enumeration*

An *enumeration* of a collection of elements is a complete listing of all the elements in it. Here's an *enumeration* of the set of values in 3..6: 3, 4, 5, 6. Here's an *enumeration* of the days of the week: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday. We can also *enumerate* the seasons: winter, spring, summer, fall.

Iterators

An *iterator* over some collection —a set or list of elements, for example— is an object that provides methods that help you write a loop to enumerate and process each element in the collection. The iterator will provide two methods:

`hasNext()`: Is there another element to enumerate?

`next()`: Return the next element to enumerate —throw an exception if there are no more.

We show you how an iterator is used. Suppose we have

```
Integer[] b= new Integer[]{5, 2, 4, 6, 3, 2};
```

and an iterator `it` that will enumerate the *even* values of `b`. We write the following loop:

```
while (it.hasNext()) {
    Integer n= it.next();
    Process n;
}
```

Each iteration of the loop enumerates another element, using the call it.next() to calculate and return it, and processes it. Execution of this loop will execute the following `Process` steps, ignoring the *odd* values in the array:

```
Process 2;  Process 4;  Process 6;  Process 2.
```

Study the above while-loop carefully. Any loop that uses an iterator to enumerate and process the values in some collection should be written exactly like this.

We show you how to write this iterator in the next video.

Interface Iterator

An iterator is a class that implements interface `java.util.Iterator`, whose documentation can be found here:

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

Besides methods `hasNext()` and `next()` shown above, Iterator has two more methods.

`Remove()` Remove the last element enumerated by this iterator. The default method throws an `UnsupportedOperationException`. We never implement this method, so if the user attempts to use it, the exception will be thrown. In many cases, it is difficult to implement properly.

`forEachRemaining(…)`: This method is outside the scope of this introduction to iterators and we will not use it.

So, we generally use only methods `hasNext()` and `next()`.