Java Iterators

Wim Bohm, Sudipto Ghosh

Motivation

- We often want to access every item in a collection of items
  - We call this ***traversing*** or ***iterating***
  - Example: array

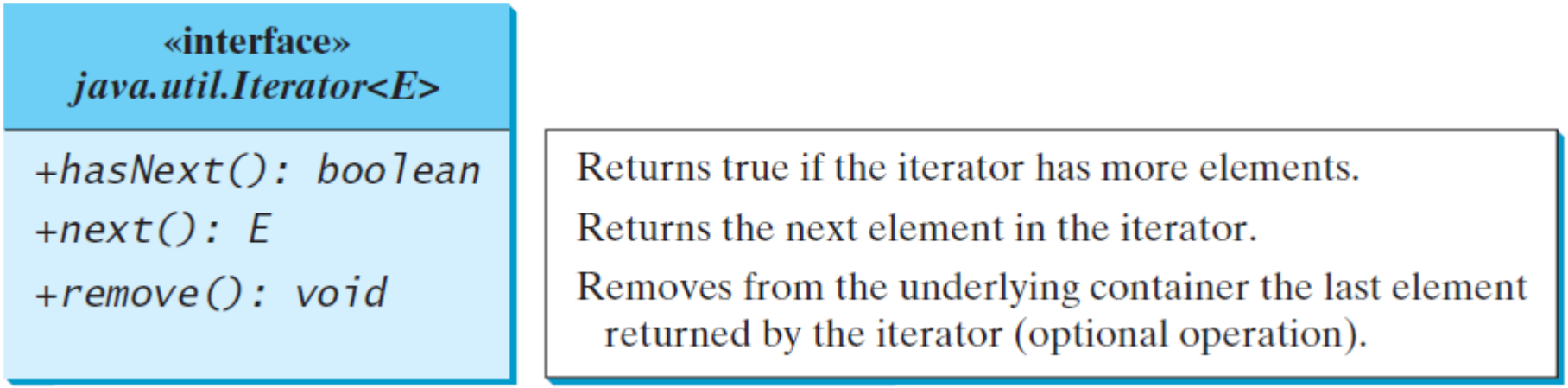
```
for (int i = 0; i < array.length; i++)
    /* do something with array[i] */
```

  - Easy because we know exactly how an array works!

Motivation

- Traversing through the elements of a collection is very common in programming, and iterators provide a *uniform* way of doing so.
- Java provides an interface for stepping through all elements in *any* collection, called an *iterator*
- Advantage? Using an iterator, we don't need to know how the data structure is implemented!

An *iterator* is an object that provides a uniform way for traversing the elements in a container such as a set, list, binary tree, etc.



Iterating through an ArrayList

- Iterating through an ArrayList of Strings:

```
for (int i = 0; i < list.size(); i++) {
   String s = list.get(i);
   //do something with s
}
```

- Alternative:

```
Iterator<String> itr = list.iterator();
while (itr.hasNext()) {
   String s = list.next();
}
```

This syntax of iteration is generic and applies to any Java class that implements the Iterator interface.

Iterating through an ArrayList

- Iterating through an ArrayList of Strings:

```
for (int i = 0; i < list.size(); i++) {
   String s = list.get(i);
   //do something with s
}
```

- Alternative:

```
Iterator<String> itr = list.iterator();
while (itr.hasNext()) {
   String s = list.next();
}
```

Advantage of the alternative: the code will work even if we decide to store the data in a different data structure (as long as it provides an iterator)

Using an iterator

```
ArrayIterator<Integer> itr = new ArrayIterator<Integer>(array);
while (itr.hasNext()){
    Integer element = itr.next();
}
```

The Iterable interface

Given an ArrayList we can traverse it using an iterator:

```
Iterator<String> itr = list.iterator();
while (itr.hasNext()) {
   String s = itr.next();
}
```

Or using the foreach form of the for loop:

```
for (String s : list) {
   //do something with s
}
```

An Iterator can only be used once.

Iterables can be the subject of “foreach” multiple times.

Possible because an ArrayList implements Iterable.

The Iterable interface

- The Java API has a generic interface called `Iterable<T>` that allows an object to be the target of a "foreach" statement
  - **`public Iterator<T> iterator();`**
    returns an iterator

- Let's check out some code: ArrayIterable.java