

Chapter 16

Iterators

Slides prepared by Rose Williams,
Binghamton University

Kenrick Mock, *University of Alaska Anchorage*

Copyright © 2017 Pearson Ltd.
All rights reserved.

1

Iterators

- An iterator is an object that is used with a collection to provide sequential access to the collection elements
 - This access allows examination and possible modification of the elements
- An iterator imposes an ordering on the elements of a collection even if the collection itself does not impose any order on the elements it contains
 - If the collection does impose an ordering on its elements, then the iterator will use the same ordering

Copyright © 2017 Pearson Ltd. All rights reserved.

16-2

2

1

The **Iterator<T>** Interface

- Java provides an **Iterator<T>** interface
 - Any object of any class that satisfies the **Iterator<T>** interface is an **Iterator<T>**
- An **Iterator<T>** does not stand on its own
 - It must be associated with some collection object using the method **iterator**
 - If **c** is an instance of a collection class (e.g., **HashSet<String>**), the following obtains an iterator for **c**:

```
Iterator iteratorForC = c.iterator();
```

Copyright © 2017 Pearson Ltd. All rights reserved.

16-3

3

Methods in the **Iterator<T>** Interface (Part 1 of 2)

Methods in the **Iterator<T>** Interface

The **Iterator<T>** interface is in the **java.util** package.
All the exception classes mentioned are the kind that are not required to be caught in a **catch** block or declared in a **throws** clause.
NoSuchElementException is in the **java.util** package, which requires an import statement if your code mentions the **NoSuchElementException** class. All the other exception classes mentioned are in the package **java.lang** and so do not require any import statement.

```
public T next()
```

Returns the next element of the collection that produced the iterator.
Throws a **NoSuchElementException** if there is no next element.

(continued)

Copyright © 2017 Pearson Ltd. All rights reserved.

16-4

4

2

Methods in the `Iterator<T>` Interface (Part 2 of 2)

Methods in the `Iterator<T>` Interface

```
public boolean hasNext()
```

Returns true if `next()` has not yet returned all the elements in the collection; returns false otherwise.

```
public void remove() (Optional)
```

Removes from the collection the last element returned by `next`.
This method can be called only once per call to `next`. If the collection is changed in any way, other than by using `remove`, the behavior of the iterator is not specified (and thus should be considered unpredictable).

Throws `IllegalStateException` if the `next` method has not yet been called, or the `remove` method has already been called after the last call to the `next` method.

Throws an `UnsupportedOperationException` if the `remove` operation is not supported by this `Iterator<T>`.

Using an Iterator with a `HashSet<T>` Object

- A `HashSet<T>` object imposes no order on the elements it contains
- However, an iterator will impose an order on the elements in the hash set
 - That is, the order in which they are produced by `next()`
 - Although the order of the elements so produced may be duplicated for each program run, there is no requirement that this must be the case

An Iterator (Part 1 of 3)

An Iterator

```
1 import java.util.HashSet;
2 import java.util.Iterator;
3
4 public class HashSetIteratorDemo
5 {
6     public static void main(String[] args)
7     {
8         HashSet<String> s = new HashSet<String>();
9
10        s.add("health");
11        s.add("love");
12        s.add("money");
13
14        System.out.println("The set contains:");
15    }
16}
```

(continued)

Copyright © 2017 Pearson Ltd. All rights reserved.

16-7

7

An Iterator (Part 2 of 3)

An Iterator

```
12     Iterator<String> i = s.iterator();
13     while (i.hasNext())
14         System.out.println(i.next());
15
16     i.remove();
17
18     System.out.println();
19     System.out.println("The set now contains:");
20
21     i = s.iterator(); ←
22     while (i.hasNext())
23         System.out.println(i.next());
24
25     System.out.println("End of program.");
26 }
```

You cannot “reset” an iterator “to the beginning.” To do a second iteration, you create another iterator.

(continued)

Copyright © 2017 Pearson Ltd. All rights reserved.

16-8

8

An Iterator (Part 3 of 3)

An Iterator

SAMPLE DIALOGUE

The set contains:
money
love
health

The HashSet<T> object does not order the elements it contains, but the iterator imposes an order on the elements.

The set now contains:
money
love
End of program.

Copyright © 2017 Pearson Ltd. All rights reserved.

16-9

9

Tip: For-Each Loops as Iterators

- Although it is not an iterator, a for-each loop can serve the same purpose as an iterator
 - A for-each loop can be used to cycle through each element in a collection
- For-each loops can be used with any of the collections discussed here

Copyright © 2017 Pearson Ltd. All rights reserved.

16-10

10

For-Each Loops as Iterators (Part 1 of 2)

For-Each Loops as Iterators

```
1 import java.util.HashSet;
2 import java.util.Iterator;
3
4 public class ForEachDemo
5 {
6     public static void main(String[] args)
7     {
8         HashSet<String> s = new HashSet<String>();
9
10        s.add("health");
11        s.add("love");
12        s.add("money");
13
14        System.out.println("The set contains:");
15    }
16}
```

(continued)

Copyright © 2017 Pearson Ltd. All rights reserved.

16-11

11

For-Each Loops as Iterators (Part 2 of 2)

For-Each Loops as Iterators

```
12     String last = null;
13     for (String e : s)
14     {
15         last = e;
16         System.out.println(e);
17     }
18
19     s.remove(last);
20
21     System.out.println();
22     System.out.println("The set now contains:");
23
24     for (String e : s)
25         System.out.println(e);
26
27     System.out.println("End of program.");
28 }
29 }  
The output is the same as in Display 16.8.
```

Copyright © 2017 Pearson Ltd. All rights reserved.

16-12

12

The `ListIterator<T>` Interface

- The `ListIterator<T>` interface extends the `Iterator<T>` interface, and is designed to work with collections that satisfy the `List<T>` interface
 - A `ListIterator<T>` has all the methods that an `Iterator<T>` has, plus additional methods
 - A `ListIterator<T>` can move in either direction along a list of elements
 - A `ListIterator<T>` has methods, such as `set` and `add`, that can be used to modify elements

Copyright © 2017 Pearson Ltd. All rights reserved.

16-13

13

Methods in the `ListIterator<T>` Interface (Part 1 of 4)

Methods in the `ListIterator<T>` Interface

The `ListIterator <T>` interface is in the `java.util` package.
The *cursor position* is explained in the text and in Display 16.11.
All the exception classes mentioned are the kind that are not required to be caught in a `catch` block or declared in a `throws` clause.
`NoSuchElementException` is in the `java.util` package, which requires an import statement if your code mentions the `NoSuchElementException` class. All the other exception classes mentioned are in the package `java.lang` and so do not require any import statement.

```
public T next()
```

Returns the next element of the list that produced the iterator. More specifically, returns the element immediately after the cursor position.
Throws a `NoSuchElementException` if there is no next element.

(continued)

Copyright © 2017 Pearson Ltd. All rights reserved.

16-14

14

Methods in the `ListIterator<T>` Interface (Part 2 of 4)

Methods in the `ListIterator<T>` Interface

```
public T previous()
```

Returns the previous element of the list that produced the iterator. More specifically, returns the element immediately before the cursor position.
Throws a `NoSuchElementException` if there is no previous element.

```
public boolean hasNext()
```

Returns `true` if there is a suitable element for `next()` to return; returns `false` otherwise.

```
public boolean hasPrevious()
```

Returns `true` if there is a suitable element for `previous()` to return; returns `false` otherwise.

```
public int nextIndex()
```

Returns the index of the element that would be returned by a call to `next()`. Returns the list size if the cursor position is at the end of the list.

(continued)

Copyright © 2017 Pearson Ltd. All rights reserved.

16-15

15

Methods in the `ListIterator<T>` Interface (Part 3 of 4)

Methods in the `ListIterator<T>` Interface

```
public int previousIndex()
```

Returns the index that would be returned by a call to `previous()`. Returns `-1` if the cursor position is at the beginning of the list.

```
public void add(T newElement) (Optional)
```

Inserts `newElement` at the location of the iterator cursor (that is, before the value, if any, that would be returned by `next()` and after the value, if any, that would be returned by `previous()`).
Cannot be used if there has been a call to `add` or `remove` since the last call to `next()` or `previous()`.
Throws `IllegalStateException` if neither `next()` nor `previous()` has been called, or the `add` or `remove` method has already been called after the last call to `next()` or `previous()`.

Throws an `UnsupportedOperationException` if the `remove` operation is not supported by this `Iterator<T>`.

Throws a `ClassCastException` if the class of `newElement` prevents it from being added.

Throws an `IllegalArgumentException` if some property other than the class of `newElement` prevents it from being added.

(continued)

Copyright © 2017 Pearson Ltd. All rights reserved.

16-16

16

Methods in the `ListIterator<T>` Interface (Part 4 of 4)

Methods in the `ListIterator<T>` Interface

```
public void remove() (Optional)
```

Removes from the collection the last element returned by `next()` or `previous()`.
This method can be called only once per call to `next()` or `previous()`.
Cannot be used if there has been a call to add or remove since the last call to `next()` or `previous()`.
Throws `IllegalStateException` if neither `next()` nor `previous()` has been called, or the add or
remove method has already been called after the last call to `next()` or `previous()`.
Throws an `UnsupportedOperationException` if the remove operation is not supported by this
`Iterator<T>`.

```
public void set(T newElement) (Optional)
```

Replaces the last element returned by `next()` or `previous()` with `newElement`.
Cannot be used if there has been a call to add or remove since the last call to `next()` or `previous()`.
Throws an `UnsupportedOperationException` if the set operation is not supported by this `Iterator<T>`.
Throws `IllegalStateException` if neither `next()` nor `previous()` has been called, or the add or
remove method has been called since the last call to `next()` or `previous()`.
Throws an `ClassCastException` if the class of `newElement` prevents it from being added.
Throws an `IllegalArgumentException` if some property other than the class of `newElement` pre-
vents it from being added.

17

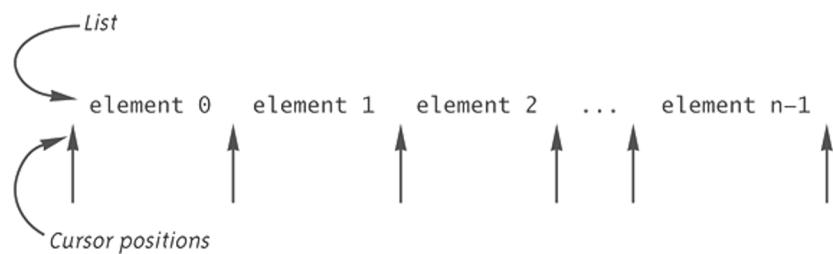
The `ListIterator<T>` Cursor

- Every `ListIterator<T>` has a position marker known as the *cursor*
 - If the list has n elements, they are numbered by indices 0 through $n-1$, but there are $n+1$ cursor positions
 - When `next()` is invoked, the element immediately following the cursor position is returned and the cursor is moved forward one cursor position
 - When `previous()` is invoked, the element immediately before the cursor position is returned and the cursor is moved back one cursor position

18

ListIterator<T> Cursor Positions

ListIterator<T> Cursor Positions



The default initial cursor position is the leftmost one.

Pitfall: **next** and **previous** Can Return a Reference

- Theoretically, when an iterator operation returns an element of the collection,
- it might return a copy or clone of the element, or it might return a reference to the element
- Iterators for the standard predefined collection classes, such as **ArrayList<T>** and **HashSet<T>**, actually return references
 - Therefore, modifying the returned value will modify the element in the collection

An Iterator Returns a Reference (Part 1 of 4)

An Iterator Returns a Reference

```
1 import java.util.ArrayList;      The class Date is defined in Display 4.13, but you can
2 import java.util.Iterator;       easily guess all you need to know about Date for this
3 public class IteratorReferenceDemo example.
4 {
5     public static void main(String[] args)
6     {
7         ArrayList<Date> birthdays = new ArrayList<Date>();
8
8         birthdays.add(new Date(1, 1, 1990));
9         birthdays.add(new Date(2, 2, 1990));
10        birthdays.add(new Date(3, 3, 1990));
11
11        System.out.println("The list contains:");
12    }
13}
```

(continued)

An Iterator Returns a Reference (Part 2 of 4)

An Iterator Returns a Reference

```
12     Iterator<Date> i = birthdays.iterator();
13     while (i.hasNext())
14         System.out.println(i.next());
15
16     i = birthdays.iterator();
17     Date d = null; //To keep the compiler happy.
18     System.out.println("Changing the references.");
19     while (i.hasNext())
20     {
21         d = i.next();
22         d.setDate(4, 1, 1990);
23     }
```

(continued)

Copyright © 2017 Pearson Ltd. All rights reserved.

16-22

22

An Iterator Returns a Reference (Part 3 of 4)

An Iterator Returns a Reference

```
23     System.out.println("The list now contains:");

24     i = birthdays.iterator();
25     while (i.hasNext())
26         System.out.println(i.next());

27     System.out.println("April fool!");
28 }
29 }
```

(continued)

Copyright © 2017 Pearson Ltd. All rights reserved.

16-23

23

An Iterator Returns a Reference (Part 4 of 4)

An Iterator Returns a Reference

SAMPLE DIALOGUE

```
The list contains:  
January 1, 1990  
February 2, 1990  
March 3, 1990  
Changing the references.  
The list now contains:  
April 1, 1990  
April 1, 1990  
April 1, 1990  
April fool!
```

Copyright © 2017 Pearson Ltd. All rights reserved.

16-24

24

Tip: Defining Your Own Iterator Classes

- There is usually little need for a programmer defined **Iterator<T>** or **ListIterator<T>** class
- The easiest and most common way to define a collection class is to make it a derived class of one of the library collection classes
 - By doing this, the **iterator()** and **listIterator()** methods automatically become available to the program
- If a collection class must be defined in some other way, then an iterator class should be defined as an inner class of the collection class