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

Adapted from material by Walter Savitch
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`:
    ```java
    Iterator iteratorForC = c.iterator();
    ```

Adapted from material by Walter Savitch
Methods in the `Iterator<T>` Interface (Part 1 of 2)

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.

```java
public T next()
```

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

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public boolean hasNext()</code></td>
<td>Returns true if <code>next()</code> has not yet returned all the elements in the collection; returns false otherwise.</td>
</tr>
<tr>
<td><code>public void remove()</code></td>
<td>Removes from the collection the last element returned by <code>next</code>. This method can be called only once per call to <code>next</code>. If the collection is changed in any way, other than by using <code>remove</code>, the behavior of the iterator is not specified (and thus should be considered unpredictable). Throws <code>IllegalStateException</code> 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 <code>UnsupportedOperationException</code> if the remove operation is not supported by this <code>Iterator&lt;T&gt;</code>.</td>
</tr>
</tbody>
</table>

Adapted from material by Walter Savitch
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

Adapted from material by Walter Savitch
An Iterator (Part 1 of 3)

An Iterator

```java
import java.util.HashSet;
import java.util.Iterator;

public class HashSetIteratorDemo {
    public static void main(String[] args) {
        HashSet<String> s = new HashSet<String>();
        s.add("health");
        s.add("love");
        s.add("money");
        System.out.println("The set contains:");
    }
}
```

(continued)
An Iterator (Part 2 of 3)

```java
An Iterator

```
```
Iterator<String> i = s.iterator();
while (i.hasNext())
    System.out.println(i.next());

i.remove();

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

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

```
```
```
i = s.iterator();
while (i.hasNext())
    System.out.println(i.next());
```
```
```
System.out.println("End of program.");
```
```
(continued)
```
An Iterator (Part 3 of 3)

Sample Dialogue

The set contains:
money
love
health

The set now contains:
money
love

End of program.

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

Adapted from material by Walter Savitch
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

Adapted from material by Walter Savitch
For-Each Loops as Iterators (Part 1 of 2)

```java
import java.util.HashSet;
import java.util.Iterator;

public classForEachDemo {
    public static void main(String[] args) {
        HashSet<String> s = new HashSet<String>();

        s.add("health");
        s.add("love");
        s.add("money");

        System.out.println("The set contains:");
```

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

```java
String last = null;
for (String e : s)
{
    last = e;
    System.out.println(e);
}

s.remove(last);
System.out.println();
System.out.println("The set now contains:");

for (String e : s)
{
    System.out.println(e);
}
System.out.println("End of program.");
```

Adapted from material by Walter Savitch
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

Adapted from material by Walter Savitch
Methods in the `ListIterator<T>` Interface

(Part 1 of 4)

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.

```java
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)
Methods in the **ListIterator<T>** Interface

(Part 2 of 4)

<table>
<thead>
<tr>
<th>Methods in the ListIterator&lt;T&gt; Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public T previous()</strong></td>
</tr>
<tr>
<td>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.</td>
</tr>
<tr>
<td><strong>public boolean hasNext()</strong></td>
</tr>
<tr>
<td>Returns true if there is a suitable element for next() to return; returns false otherwise.</td>
</tr>
<tr>
<td><strong>public boolean hasPrevious()</strong></td>
</tr>
<tr>
<td>Returns true if there is a suitable element for previous() to return; returns false otherwise.</td>
</tr>
<tr>
<td><strong>public int nextIndex()</strong></td>
</tr>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

(continued)

Adapted from material by Walter Savitch
Methods in the `ListIterator<T>` Interface

(continued)
Methods in the `ListIterator<T>` Interface

(Part 4 of 4)

### 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 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.

Adapted from material by Walter Savitch
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

Adapted from material by Walter Savitch
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

Adapted from material by Walter Savitch
An Iterator Returns a Reference (Part 1 of 4)

```java
import java.util.ArrayList;
import java.util.Iterator;

public class IteratorReferenceDemo {
    public static void main(String[] args) {
        ArrayList<Date> birthdays = new ArrayList<Date>();

        birthdays.add(new Date(1, 1, 1990));
        birthdays.add(new Date(2, 2, 1990));
        birthdays.add(new Date(3, 3, 1990));

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

The class Date is defined in Display 4.13, but you can easily guess all you need to know about Date for this example.

Adapted from material by Walter Savitch
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     i = birthdays.iterator();
16     Date d = null;  //To keep the compiler happy.
17     System.out.println("Changing the references.");
18     while (i.hasNext())
19     {
20         d = i.next();
21         d.setDate(4, 1, 1990);
22     }
```

(continued)

Adapted from material by Walter Savitch
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)
An Iterator Returns a Reference (Part 4 of 4)

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 1, 1990
April fool!

Adapted from material by Walter Savitch
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

Adapted from material by Walter Savitch