Container Traversal

CMPUT 115 - Lecture 17
Department of Computing Science
University of Alberta

Some code in this lecture is based on code from the book
Java Structures by Duane A. Bailey or the companion structure package.

About This Lecture

- In this lecture we will learn about traversing containers.
- In Java, traversal is commonly done using the Enumeration and Iterator Interfaces.

Outline

- Traversal
- Cursors
- The Enumeration Interface
- The Iterator Interface
- Vector Iterators
- Singly-Linked List Iterators

Container Traversal

- A traversal is an operation that accesses each element of a container once and performs some operation on the element as it is accessed.
- For example, to print out all elements in a container, traverse the container and call System.out.println with each element.
- If a container is modified while traversing it, unexpected results may occur.

Array Traversal

- If the container is an array (or Vector) traversal can be done with a simple loop:

```java
// initialize
int index = 0;
// while there are more elements to process
while (index < array.length) {
    // get the current element and move to the next
    element = array[index++];
    // process the current element
    System.out.println(element);
}
```

List Traversal – inside the List class

- A similar loop can be used to traverse a linked list, BUT ONLY IF we are writing code inside the List class.

```java
// initialize
SinglyLinkedListElement index = head;
// while there are more elements to process
while (index != null) {
    // get the current element and move to the next
    element = index.data;
    index = index.next;
    // process the current element
    System.out.println(element);
}
```
Approaches to Traversal

- What could we add to the List class (or any other Container class) to enable traversal with a similar loop to be written OUTSIDE the List class?
- We will look at two approaches:
  - Cursors
  - Iterators

Cursor Operations (1)

- A cursor can be added to a Container to enable traversals to be written outside the Container class.
- A cursor is something that can be moved across the elements of the container one at a time.

```
public Object nextElement() {
    // pre: the cursor has not yet returned all the
    //   container's elements
    // post: returns the element the cursor is currently
    //   pointing at and moves the cursor to the next
    //   element
```

Cursor Operations (2)

- To test if the cursor has passed over all the container's elements:

```
public boolean hasMoreElements() {
    // post: true unless the cursor has returned all the
    //   elements in the container
```

- To reset the cursor so that it can be used to look at all the elements again:

```
public void reset() {
    // post: makes the cursor ready for a new traversal
```

A Singly-Linked List with a Cursor

Typically a cursor is implemented as an extra instance variable that points to an element in the Container.

Container Traversal with a Cursor

```
// initialize
aContainer.reset();
// while there are more elements to process
while (aContainer.hasMoreElements()) {
    // get the current element and move to the next
    element = aContainer.nextElement();
    // process the current element
    System.out.println( element );
}
```

- Note that the messages are being sent to the container, the cursor is not an external object capable of receiving messages.

Iterators

- The main drawback of the cursor approach is that the cursor is an intrinsic part of the Container. If additional cursors are needed, they can only be created by changing the code that implements the Container.
- Iterators overcome this deficiency. An Iterator gives the same functionality as a cursor, but Iterators are created dynamically.
  - Dynamic cursors: more than one cursor may be created and created whenever needed
  - Instead of adding a cursor to the Container class, we add a method to create an Iterator.

```
public Iterator elements();
    // post: return an iterator for traversing the container
```
Container Traversal with an Iterator

// initialize
Iterator it = container.elements();
// while there are more elements to process
while (it.hasMoreElements()) {
// get the current element and move to the next
element = it.nextElement();
// process the current element
System.out.println(element);
}

- Note that the messages are being sent to the iterator, not the container

Iterators and Enumerations

- The structure package has two interfaces to support iterator-style traversals: Enumeration and Iterator.
- The Enumeration Interface has the minimal set of methods
  - hasMoreElements()
  - nextElement()
- The Iterator Interface adds methods
  - reset()
  - value() // peeks ahead without moving

Structure Interface - Enumeration

```java
public interface Enumeration {
    public boolean hasMoreElements();
    // post: returns true iff at least one element of the
    // associated container has yet to be enumerated.

    public Object nextElement();
    // pre: the associated container has more
    // unenumerated elements.
    // post: returns an unenumerated element of the
    // associated container and marks it as enumerated.
    // There is no fixed order. Two Enumerations for the
    // same container might order the elements differently.
}
```

Structure Interface - Iterator

```java
public interface Iterator extends Enumeration {
    public Object nextElement();
    // pre: the associated container has more
    // uniterated elements
    // post: returns the next uniterated element of the
    // associated container and marks it as iterated
    // There is a fixed order. Two Iterators for the
    // same container must order the elements the same.

    public void reset();
    // post: Makes the Iterator ready for a new traversal

    public Object value();
    // pre: the associated container has more
    // uniterated elements
    // post: Returns the next element to be iterated
}
```

Enumerations and Iterators in java.*

- The java.* packages also have Enumeration and Iterator Interfaces, but:
  - Iterator does not extend Enumeration (as it does in the structure package)
  - The Iterator Interface has different methods in the java.* packages than in the structure package

Iterator Example – illustrating value()

```java
public static void main(String[] args) {
    Vector container;
    Iterator it;
    container = new Vector();
    container.addElement("Fred");
    container.addElement("Barney");
    it = container.elements();
    while (it.hasMoreElements()) {
        System.out.println(it.value());
    }
}
```

Output

<table>
<thead>
<tr>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
</tr>
<tr>
<td>Fred</td>
</tr>
<tr>
<td>Barney</td>
</tr>
<tr>
<td>Barney</td>
</tr>
</tbody>
</table>
VectorIterator

- “it” in the previous example is an instance of some class implementing the iterator interface. Let’s call this class VectorIterator, since it will define a type of iterator that is specifically for the Vector class.
- VectorIterator must implement the methods in the iterator interface, and it must have instance variables which allow a VectorIterator instance to keep track of its current state:
  - which instance of Vector is it iterating over?
  - what position in this Vector instance is it currently at?

```java
class VectorIterator implements Iterator {
    protected Vector vector;
    protected int current;

    public VectorIterator(Vector v) {
        // post: initializes Vector and resets the traversal
        this.vector = v;
        this.reset();
    }

    public void reset() {
        // post: Makes the Iterator ready for a new traversal
        this.current = 0;
    }

    public Object value() {
        // pre: the associated container has more
        // uniterated elements
        // post: Returns the next element to be iterated
        return this.vector.elementAt(this.current);
    }

    public boolean hasMoreElements() {
        // post: returns true iff at least one element of the
        // associated container has yet to be enumerated.
        return this.current < this.vector.size();
    }

    public Object nextElement() {
        // pre: the associated container has more
        // uniterated elements
        // post: returns the next uniterated element of the
        // associated container and marks it as iterated
        Object element;
        element = this.vector.elementAt(this.current);
        this.current++;
        return element;
    }
}
```

VectorIterator - Iterator Interface

```java
/* Interface Iterator Methods */
public void reset() { // post: Makes the Iterator ready for a new traversal
    this.current = 0;
}

public Object value() { // pre: the associated container has more
    // uniterated elements
    // post: Returns the next element to be iterated
    return this.vector.elementAt(this.current);
}
```

VectorIterator - Enumeration Interface

```java
/* Interface Enumeration Methods */
public boolean hasMoreElements() { // post: returns true iff at least one element of the
    // associated container has yet to be enumerated.
    return this.current < this.vector.size();
}

public Object nextElement() { // pre: the associated container has more
    // uniterated elements
    // post: returns the next uniterated element of the
    // associated container and marks it as iterated
    Object element;
    element = this.vector.elementAt(this.current);
    this.current++;
    return element;
}
```

Class Vector - elements()

- With VectorIterator defined we can now write the elements() method in the Vector class.

```java
public Iterator elements() { // post: returns an Iterator for traversing the elements
    return new VectorIterator(this);
}
```

Where to place Iterator classes?

- Until now, all the classes we have defined have been placed in files of their own, but this is not what to do with classes implementing iterators.
- An iterator often needs to access private information about the data structure it is iterating over (this is not the case for VectorIterator but it is true for most iterators).
- For that reason classes defining an Iterator are placed inside the same file as the class they iterate over: the VectorIterator class would be placed in the Vector.java file along with the Vector class.
**Hidden Classes**

- Putting the classes implementing Iterator in the same file as the Collection class they iterate over makes them hidden – they are not public.
- Therefore:
  - The user cannot call the constructor of the class implementing Iterator. The only way the user can create an instance of an Iterator class is indirectly, by calling the `elements()` method of the associated Collection class.
  - The user cannot know the name of class Object returned by `elements()`. The user only knows that the Object implements the Iterator interface.

**Caution when Using Iterators**

- Never modify a data structure while an associated Iterator is in the midst of its traversal.
- Treat the values returned by Iterators as read-only.

**Iterator Example – SinglyLinkedList**

```java
public static void main(String[] args) {
    SinglyLinkedList container;
    Iterator it;
    container = new SinglyLinkedList();
    container.addToTail("Fred");
    container.addToTail("Barney");
    it = container.elements();
    while (it.hasMoreElements()) {
        System.out.println(it.value());
        System.out.println(it.nextElement());
    }
}
```

**Output**

Fred
Fred
Fred
Barney
Barney

**SinglyLinkedList.java**

```java
public class SinglyLinkedList implements List {
    // definition of the SinglyLinkedList class
} // end of the SinglyLinkedList class definition

class SLLIterator implements Iterator {
    // definition of the SLLIterator class
} // end of the SLLIterator class definition
```

**SLLIterator – State and Constructor**

```java
class SLLIterator implements Iterator {
    // Iterator for SinglyLinkedLists
    protected SinglyLinkedListElement head;
    protected SinglyLinkedListElement current;

    public SLLIterator(SinglyLinkedListElement head) {
        // post: save the starting point, reset the traversal
        this.head = head;
        this.reset();
    }
```
SLLIterator - Iterator Interface

/* Interface Iterator Methods */
public void reset() {
    // post: Makes the Iterator ready for a new traversal
    this.current = head;
}

public Object value() {
    // pre: the associated container has more
    // uniterated elements
    // post: Returns the next element to be iterated
    return this.current.data;
}

SLLIterator - Enumeration Interface

/* Interface Enumeration Methods */
public boolean hasMoreElements() {
    // post: returns true iff at least one element of the
    // associated container has yet to be enumerated.
    return this.current != null;
}

public Object nextElement() {
    // pre: the associated container has more
    // uniterated elements
    // post: returns the next uniterated element of the
    // associated container and marks it as iterated
    Object element;
    element = this.current.data;
    this.current = this.current.next;
    return element;
}

Class SinglyLinkedList - elements()

public Iterator elements() {
    // post: returns an Iterator for traversing the elements
    return new SLLIterator(this.head);
}