

Topic 13

Iterators

9-1

Motivation

- We often want to access every item in a data structure or collection in turn
 - We call this **traversing** or **iterating over** or **stepping through** or **visiting every item in** the data structure or collection
- Example with a data structure (array):

```
for (int i = 0; i < arr.length(); i++)  
    /* do something to arr[i] */
```
- This is straightforward because we know exactly how an array works!

9-2

Motivation

- What if we want to traverse a **collection** of objects?
 - A list, a stack, a queue ...
 - Its underlying implementation may not be known to us
- Java provides a **common scheme** for stepping through all elements in **any** collection, called an **iterator**

9-3

What is an Iterator?

- An **iterator** is a mechanism used to step through the elements of a collection one by one
 - Each element is "**delivered**" exactly once
- **Example**
 - Iterate through an ordered list and print each element in turn



9-4

2-4

Iterator Interface

- The Java API has a generic **interface** called **Iterator<T>** that specifies what methods are required of an iterator
 - **public boolean hasNext();**
returns true if there are more elements in the iteration
 - **public T next();**
returns the next element in the iteration
 - **public void remove();**
removes the last element returned by the iterator (*optional operation*)
- It is in the **java.util** package of the Java API

9-5

2-5

Array Iterator

- If we had a collection with an array implementation, we would need an **array implementation** of the **Iterator** interface
 - See **ArrayIterator.java**:
 - Its attributes
 - Its constructor
 - The code for the methods **hasNext** and **next**
 - In what order does it deliver the items?
- **Note:** **ArrayIterator.java** can be used by an array implementation of **any** collection!

9-6

```
// Represents an iterator over the elements of an array
```

```
import java.util.*;
```

```
public class ArrayIterator<T> implements Iterator<T> {
```

```
    // Attributes
```

```
    private int count; // number of elements in collection
```

```
    private int current; // current position in the iteration
```

```
    private T[] items; // items in the collection
```

```
    // Constructor: sets up this iterator using the  
    // specified items
```

```
    public ArrayIterator (T[] collection, int size) {
```

```
        items = collection;
```

```
        count = size;
```

```
        current = 0;
```

```
    }
```

```
    // cont'd..
```

ArrayIterator.java

2-7

```
    // cont'd..
```

```
    // Returns true if this iterator has at least one  
    // more element to deliver in the iteration
```

```
    public boolean hasNext() {  
        return (current < count);  
    }
```

```
    // Returns the next element in the iteration.  
    // If there are no more elements in this iteration,  
    // throws an exception.
```

```
    public T next() {
```

```
        if (!hasNext())
```

```
            throw new NoSuchElementException();
```

```
        current++;
```

```
        return items[current - 1];  
    }
```

```
}
```

ArrayIterator.java (cont'd)

9-8

2-8

Linked Iterator

- If we had a collection with a linked implementation, we would need a **linked implementation** of the **Iterator** interface
 - See **LinkedListIterator.java**
 - Its attributes
 - Its constructor
 - The code for the methods **hasNext** and **next**
 - In what order does it deliver the items?
- **Note:** **LinkedListIterator.java** can be used by a linked implementation of **any** collection!

9-9

```
import java.util.*;  
public class LinkedListIterator<T> implements Iterator<T> {
```

```
    // Attributes
```

```
    private int count; // number of elements in collection
```

```
    private LinearNode<T> current; // current position
```

```
    // Constructor: Sets up this iterator using the specified items
```

```
    public LinkedListIterator (LinearNode<T> collection, int size){
```

```
        current = collection;
```

```
        count = size;
```

```
    } //cont'd..
```

LinkedListIterator.java

9-10

```
// ..cont'd..
```

```
// Returns true if this iterator has at least one more element  
// to deliver in the iteration.
```

```
public boolean hasNext() {  
    return (current!= null);  
}
```

```
// Returns the next element in the iteration. If there are no  
// more elements in this iteration, throws an exception.
```

```
public T next() {
```

```
    if (!hasNext())
```

```
        throw new NoSuchElementException();
```

```
    T result = current.getElement();
```

```
    current = current.getNext();
```

```
    return result;
```

```
}
```

**LinkedListIterator.java
(cont'd)**

9-11

Iterators for a Collection

So how do we set up an iterator for a collection?

- Recall that the ListADT interface has an **operation** called **iterator** :

```
// Returns an iterator for the elements in this list
```

```
public Iterator<T> iterator();
```

- (In fact, any of our collections could have had an **iterator** operation ... later)

9-12

The iterator Operation in the ListADT

- Note that the return type of the iterator operation is `Iterator<T>`
 - But `Iterator<T>` is an interface, not a class!
 - When the return type of a method is an *interface name*, the method actually returns an object from *a class that implements the interface*
 - The iterator operation in `ArrayList` will use the class `ArrayIterator`
 - The iterator operation in `LinkedList` will use the class `LinkedListIterator`

9-13

iterator method for ArrayList

```
/**
 * Returns an iterator for the elements currently in this list.
 *
 * @return an iterator for the elements in this list
 */
public Iterator<T> iterator()
{
    return new ArrayIterator<T>(list, rear);
}
```

9-14

2-14

iterator method for LinkedList

```
/**
 * Returns an iterator for the elements currently in this list.
 *
 * @return an iterator for the elements in this list
 */
public Iterator<T> iterator()
{
    return new LinkedListIterator<T>(contents, count);
}
```

The only difference from the iterator method in `ArrayList` is the class from which the iterator object is being created!

9-15

Using an Iterator

- When the `iterator()` method in a collection is invoked, it returns an “iterator object”
- We can then invoke the methods `hasNext()` and `next()` on that object, to iterate through the collection
 - (Those are the methods that are specified in the `Iterator<T>` interface)

9-16

Using an Iterator in an Application

Example: Suppose we had an unordered list that was created by

```
ArrayUnorderedList<Person> myList =
    new ArrayUnorderedList<Person>();
```

and then had items added to it...

```
// Use iterator to display contents of list
Iterator<Person> iter = myList.iterator();
while(iter.hasNext())
{
    System.out.println(iter.next());
}
```

// cont'd

9-17

Using an Iterator in an Application

```
// Print just the email addresses now
```

```
// Note that we have to start a new iteration!
```

```
iter = myList.iterator(); // start new iteration
while(iter.hasNext())
{
    System.out.println(iter.next().getEmail());
}
```

9-18

Example: Using an Iterator within a Class Definition

- Rewrite the toString() method of ArrayList using its iterator:

```
public String toString() {
    String result = "";

    Iterator<T> iter = this.iterator();

    while ( iter.hasNext() )
        result = result + iter.next().toString() + "\n";

    return result;
}
```

9-19 2-19

Discussion

- Could we use the *very same code* from the previous slide for the toString() method of LinkedList?
- If we had an **iterator** operation in the StackADT, could we use this very same code for the toString() methods of the StackADT implementations?

9-20

Exercises

- Add an **iterator** operation to the StackADT
 - Implement it in `ArrayStack`
 - In what order will it deliver the items if we use `ArrayIterator.java` to implement the `Iterator<T>` interface?
 - Implement it in `LinkedStack`
 - In what order will it deliver the items if we use `LinkedListIterator.java` to implement the `Iterator<T>` interface?
 - Rewrite the `toString` method of the StackADT implementations to use its iterator
- Ditto for the QueueADT

9-21

Discussion

- Note that the order of the iteration is determined by the **design of the class that implements the `Iterator<T>` interface**
- If we wanted an iterator that delivered the items in a stack in the opposite order from `ArrayIterator`, what would we have to do?

9-22

Why use Iterators?

- Traversing through the elements of a collection is very common in programming, and iterators provide a *uniform* way of doing so
- **Advantage? Using an iterator, we don't need to know how the collection is implemented!**

9-23 2-23