Lecture 11: Iterators

CS2504/CS4092– Algorithms and Linear Data Structures

Dr. Kieran T. Herley

Department of Computer Science
University College Cork

2013/14

Iterator Concept

- A common theme in CS is the idea of "walking through" a set of elements in order to perform some action.
- Examples:
  - Generating all the prime numbers from one to a million
  - Printing out the contents of an array
  - Searching through a linked list to find a particular element
- Java provides the concept of an *iterator* to to abstract this idea

Java Iterator

Java supports the interface java.util.Iterator<E> that allows us to "walk" forwards through a set of elements one by one. The "position" of the "walk" is marked by a "cursor" that is initially before the first element and advanced one element at a time. The interface includes the following operations.

hasNext(): Return true if there are one or more elements in front of the cursor. *Input: None*; *Output: boolean*.

next(): Return the element immediately in front of the cursor and advance the cursor past this item. Illegal if cursor is at the end of the collection. *Input: None*; *Output: E*.

Using an Iterator

PrimeNumbers.java Class that implements the java.util.Iterator interface; supports "walk" through the first *n* prime numbers

Usage

```
PrimeNumbers primes = new PrimeNumbers(10);

while (primes.hasNext())
{   System.out.println(primes.next());
}
```

Outcome Prints the first 10 primes

Method iterator() of ADT List

In addition to operations mentioned earlier, ADT List supports the following additional operation

> **iterator():** Return an iterator of the elements of this list. *Input: None*; *Output: Iterator<E>*.

Using Method iterator

Setting Suppose lst is List of Integer elements

Usage

```
Iterator<Integer> lstIter = lst.iterator();
while (lstIter.hasNext())
{  System.out.println(lstIter.next());
}
```

Note

Foreach Loop

Setting lst as before

Foreach Loop Can simplify iteration process using variation of Java for loop (works with Java 1.5 an later only)

Usage

```
for (Integer num: lst)
{  System.out.println(num);
}
```

Semantics Variable num takes on each element of lst in turn; for each element the body of the loop is executed.

Technical Note ADT must support special java.util.Iteratable interface (as ADT List does) for this to work

Foreach Loop cont'd

foreach

```
for (Integer num: lst)
{   System.out.println(num);
}
```

Equvalent for Loop

```
for (Iterator<Integer> it = lst.iterator(); it.has
{   Integer num = it.next();
    System.out.println(num);
}
```

Implementing Iterator Concept

List Interface

```
public interface List<EltType>
        extends java.lang.Iterable<EltType>
{   . . .
    public Iterator<EltType> iterator();
}
```

List Implementation

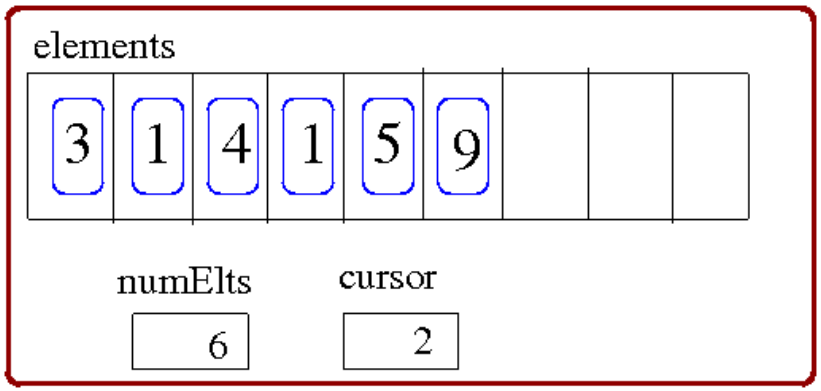
```
public class ArrayBasedList<EltType>
        implements List<EltType>
{   . . .
    public Iterator<EltType> iterator(){. . .}
}
```

Both import java.util.Iterator

Impl'ing Iterator– Simple Approach

ArrayBasedIterator Implement special class to hold "snaphot" of current state of the list.

- Uses left-justified array to hold (copies of) elements and int to represent "cursor" position
- Support Iterator methods hasNext and next
- Supports (non-Iterator) method add (appends) to populate iterator with contents initially



Impl'ing Iterator– Simple Approach

The iterator() method of ArrayBasedList]:

```
public  Iterator <EltType> iterator()
{   ArrayBasedIterator<EltType> it = new ArrayBasedIterator<EltType>()

    for (int i = 0; i < numElts; i++)
       it .add(elements[i ]);
    return it ;
}
```

NB: add is not an Iterator method. It used during the "construction phase" to populate the iterator prior to use.

Using the Iterator

Suppose that *lst* is list of type List<String>

```
Iterator it<String> = lst.iterator();
while (it.hasNext())
{   it.next();
}
```

or could use foreach construct.

Impl'ing Iterator–Better Approach

- "Snapshot" approach assumes that underlying list will not change while Iterator object returned by method iterator() is in use; also creating complete copy may be expensive.
- Another approach:
  - Interact directly with actual list itself
  - Create object that embodies a "position" within underlying list that is interrogated/updated in response to hasNext/next operations

ABLIterator

```
public class ABLIterator<EltType>
             implements Iterator<EltType>
{  public ABLIterator(ArrayBasedList<EltType> l)
   {  lst = l; cursor = 0; }
   . . .
   protected ArrayBasedList<EltType> lst;
   protected int cursor;
}
```

ABLIterator

```
public boolean hasNext()
{   return (cursor < lst.numElts);
}

public EltType next()
{   . . .
    return lst.elements[cursor++];
}
```

Method iterator() in ArrayBasedList

```
public class ArrayBasedList<EltType>
             implements List<EltType>
{  . . .
   public Iterator <EltType> iterator()
   {  return new ABLIterator<EltType>(this);}
}
```

Technical Note

- Making ArrayBasedList's instance variables protected ensures that they are visible to classes in the same "package" including ABLIterator
-