Iterating over a list

SinglyLinkedList<

type
/> list =
   new SinglyLinkedList<


type
/>();

...

for (int i=0; i<list.size(); i++) {
   type nextElement = list.get(i);
   // do something with nextElement
}

If the list has n elements, what is the
order of complexity of this iteration? ________
Iterators

• An iterator implements a "marker" in the list to keep track of the last element accessed so we can examine the next element quickly.

```
Iterator<type> iter = list.iterator();
while (iter.hasNext()) {
    type nextElement = iter.next();
    // do something with nextElement
}
```

If the list has n elements, what is the order of complexity of this iteration? ________
The Iterator interface

boolean hasNext()
Returns true if there is another element to process.

E next()
Returns the next element. If there are no more elements, throws the NoSuchElementException.

void remove()
Removes the last element returned by the next method. (must be preceded by a call to next)

Using Iterator interface

• If we say SinglyLinkedList implements Iterator, we can only have one iterator for the list.
  • That is, the singly linked list acts as the iterator itself.
• Instead, we can create an iterator as an inner class.
  • We can have more than one iterator for a list.
Creating iterators

To use an iterator on a collection of data, we must supply an iterator method that returns an Iterator on this object.

Example: in SinglyLinkedList class

```java
public Iterator<E> iterator() {
    return new SLLIterator();
}
```

Using iterators

Example: Use an iterator to remove all integers in a singly-linked list of integers that are even.

```java
SinglyLinkedList<Integer> list =
    new SinglyLinkedList<Integer>();
...
Iterator<Integer> iter = list.iterator();
while (iter.hasNext()) {
    int num = iter.next();
    if (num % 2 == 0) iter.remove();
}
The `ListIterator<E>` interface

- `ListIterator` is an extension of `Iterator`.
- The `LinkedList` class implements the `List<E>` interface using a doubly-linked list.
- Methods in `LinkedList` that return a list iterator:
  
  ```java
  public ListIterator<E> listIterator()
  public ListIterator<E> listIterator(int index)
  ```
- Methods in the `ListIterator` interface:
  - `add`, `hasNext`, `hasPrevious`, `next`, `previous`, `nextIndex`, `previousIndex`, `remove`, `set`

Example

- Replace the first occurrence of `target` in `LinkedList` `aList` of strings with `newItem`:

  ```java
  ListIterator<String> iter = aList.listIterator();
  while (iter.hasNext()) {
      if (target.equals(iter.next())) {
          iter.set(newItem);
          break;
      }
  }
  ```
Example

Count the number of times target appears in LinkedList aList of strings:

```java
int count = 0;
ListIterator<String> iter = aList.listIterator();
while (iter.hasNext()) {
    if (target.equals(iter.next())) {
        count++;
    }
}
```

Example (using the enhanced for loop)

Count the number of times target appears in LinkedList aList of strings:

```java
int count = 0;
for (String nextStr : aList) {
    if (target.equals(nextStr)) {
        count++;
    }
}
```
Example
(using the enhanced for loop)

- Enhanced for loops can also be used with arrays.

```java
int[] dataArray = new int[1000];
...
int total = 0;
for (int nextInt : dataArray) {
    total += nextInt;
}
```

Iterable<T> interface

- Specifies an iterator method.
  - Iterator<T> iterator()
    Returns an iterator over a set of elements of type T.
- Implemented by the Collection interface.
- All classes that implement the Collection interface must include an iterator method that returns an Iterator for that collection.
- The enhanced for statement can then be used to "traverse" the collection one element at a time easily.
Example

- Let myList be an ArrayList of Integer.
- Since myList is an ArrayList, and ArrayList is a subclass of Collection, it must have an iterator method that returns an iterator for the collection.

```java
int total = 0;
for (int nextInt : myList)
    total += nextInt;
```

Implementing an iterator

Inside the SinglyLinkedList class:
```java
private class SLLIterator implements Iterator<E>
{
    private Node<E> nodePtr;
    private Node<E> prevPtr;
    private Node<E> prev2Ptr;
    private boolean okToRemove;
    ...
    // constructor and
    // required methods
}
```
Implementing an iterator

Constructor for the SLLIterator:

```java
private SLLIterator() {
    nodePtr = head;
    prevPtr = null;
    prev2Ptr = null;
    okToRemove = false;
}
```

We can only remove if we call next first.

Implementing an iterator

Required methods for SLLIterator:
hasNext, next, remove

```java
public boolean hasNext() {
    return nodePtr != null;
}
```
Implementing an iterator

public E next() {
    if (nodePtr == null)
        throw new NoSuchElementException();
    E result = nodePtr.data;
    prev2Ptr = prevPtr;
    prevPtr = nodePtr;
    nodePtr = nodePtr.next;
    okToRemove = true;
    return result;
}

Implementing an iterator

public void remove() {
    if (!okToRemove)
        throw new IllegalStateException();
    if (prev2Ptr == null)
        head = nodePtr;
    else
        prev2Ptr.next = nodePtr;
    prevPtr = prev2Ptr;
    okToRemove = false;
}

Does this method work if you remove the tail node?

What if the list only has 1 node?
Properties of All Collections

- Collections grow in size as needed.
- Collections always hold references to objects.
- Collections must have at least two constructors:
  - A constructor to create an empty collection (no parameters).
  - A constructor to make a copy of another collection (one parameter of type Collection).
AbstractCollection

- The AbstractCollection class helps us define a collection.
- This abstract class contains implementations of most methods except size and iterator.
- To create a collection, we only have to extend this class, provide methods for size and iterator, and include an inner class to implement the Iterator interface.

```java
public class SinglyLinkedList<E>
    extends AbstractCollection<E> {...}
```
- From the Java API: To implement a modifiable collection, the programmer must additionally override this class's add method (which otherwise throws an UnsupportedOperationException), and the iterator returned by the iterator method must additionally implement its remove method.

AbstractList

- The AbstractList class helps us define a list.
- This abstract class contains implementations of most methods except add, get, remove, set, and size.
- To create a list, we only have to extend AbstractList, provide the missing methods above.
  - used for collections that can be accessed randomly
  - how does it return an iterator?

```java
public class CMUArrayList<E>
    extends AbstractList<E> {...}
```
AbstractSequentialList

- The AbstractSequentialList class helps us define a sequential list.
- This abstract class contains implementations of most methods except listIterator and size, and provide an inner class that implements the ListIterator interface.
- To create a list, we only have to extend AbstractListCollection, provide the missing methods above.
  - used for collections accessed sequentially