Example: LinkedStack<T>

- LinkedStack UML Class Diagram
- LinkedStack<T> Class
- LinkedStack Attributes/Constructor
- LinkedStack Methods
- LinkedStack iterator method
- LinkedIterator<T> class
- Project 1 Q/A
- Reading: L&C 7.4
LinkedStack UML Class Diagram

LinkedStack<T>
- count : int
- top : LinearNode<T>
+ LinkedStack()

LinearNode<T>
- next : LinearNode<T>
- element : T
+ LinearNode(void)
+ LinearNode(element : T)
+ setNext(node : void)
+ getNext( void) : LinearNode
+ getElement( ) : T
+ setElement(element : T) : void

StackADT<T>
+ push(element : T) : void
+ pop() : T
+ peek() : T
+ isEmpty() : T
+ size() : T
+ toString() : T

LinkedIterator<T>

Iterator<T>

Iterable<T>
+ iterator() : Iterator<T>

Self referential or Recursive Assoc.
LinkedStack<T> Class

• The LinkedStack<T> class implements the same StackADT<T> interface as ArrayStack<T> using a singly linked list instead of an array inside.

• Externally, your code does not need to be much different when using the LinkedStack<T> class instead of the ArrayStack<T> class:

```java
StackADT<String>myStack = new ArrayStack<String>();
OR
    = new LinkedStack<String>();
```

• Internally, method code of the LinkedStack<T> class is very different due to the difference in the underlying data structure being used.
LinkedStack<T> Class

- Again unlike the text author: We assume that `StackADT<T> interface` extends `Iterable<T>`
- If not, we’d need to add `Iterable<T>` after `StackADT<T>` in the implements clause of the `LinkedStack<T> class` header
LinkedStack Attributes/Constructor

• The class definition starts with:

```java
public class LinkedStack<T>
    implements StackADT<T>
{
    private int count;
    private LinearNode<T> top;

    public LinkedStack() // default constructor
    {
        count = 0;
        top = null; // NOTE: not an array
    }
```
LinkedStack Attributes/Constructor

• The variable “top” is an object reference variable – not an array
• We don’t need to define any default capacity or pass an initial capacity to the constructor for the size of an array
• We don’t need to instantiate an array object or any LinearNode objects in the constructor – just set top to null for a null terminated linked list
LinkedStack Methods

• Note that because we are not using a fixed size data structure such as an array, we don’t need a private `expandCapacity()` method for the `push` method.

• Again, these lecture notes may do some code differently from the textbook so that you can see more than one way of doing it.
LinkedStack Methods

• push – O(1)

```java
public void push (T element) {
    LinearNode<T> node = new LinearNode<T>(element);
    node.setNext(top);
    top = node;
    count++;
}
```
LinkedStack Methods

• Pop - O(1)

```java
public T pop() throws EmptyCollectionException
{
    if (isEmpty())
        throw new EmptyCollectionException("stack");
    T result = top.getElement();
    top = top.getNext();  // creates garbage
    count--;

    return result;
}
```
LinkedStack Methods

• Peek – O(1)

```java
public T peek()
{
    if (isEmpty())
        throw new EmptyCollectionException("stack");
    return top.getElement();
}
```
LinkedStack iterator Method

• iterator – O(1)
  
  ```java
  public Iterator<T> iterator
  {
    return new LinkedIterator<T>(contents);
  }
  ```

• We will study the LinkedIterator class to understand how it is implemented

• Again, it makes sense to implement the LinkedIterator class as an “inner class”
LinkedIterator<T> Class

• The iterator method of the LinkedStack class instantiates and returns a reference to a LinkedIterator object to its caller

• The LinkedIterator code can access the outer class’s reference to the first LinearNode<T> object for this instance of the linked structure
LinkedIterator<T> Class

• Class / Attribute Definitions and Constructor

```java
public class LinkedIterator<T> implements Iterator<T>
{
    private LinearNode<T> current; // current position

    public LinkedIterator()
    {
        current = top;
    }
}
```
LinkedIterator Methods

• hasNext – O(1)

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

• next – O(1)

    public T next()
    {
        if (!hasNext())
            throw new NoSuchElementException();
        T result = current.getElement();
        current = current.getNext();
        return result;
    } // old LinearNode does not become garbage (Why?)
LinkedIterator Methods

• remove – O(1)
• We don’t need to implement real code for the remove method, but there is no return value that we can use to indicate that it is not implemented
• If we don’t implement it, we indicate that the code is not implemented by throwing an exception

```java
public void remove() throws UnsupportedOperationException {
    throw new UnsupportedOperationException();
}
```
LinkedIterator Methods

• Again, if we implement the remove method, notice that we don’t specify the element that is to be removed and we do not return a reference to the element being removed

• It is assumed that the calling code has been iterating on condition hasNext() and calling next() and already has a reference

• The last element returned by next() is the element that will be removed
LinkedIterator Method Analysis

• Each of the LinkedIterator methods is O(1)
• However, they are usually called inside an external while loop or “for-each” loop
• Hence, the process of “iterating” through a collection using an Iterator is O(n)
LinkedLIstIterator Class in Textbook

• Again, the textbook iterator class detects any modification to the linked list and causes the iteration to “fast-fail” with an exception
• I did not show that in my example code
Project 1 Q/A

- Project 1 is due before next class
- Any questions about Project 1?