In this lecture we will learn about an implementation of the List Interface called Circular List.

Outline
- Drawing Circular Lists
- CircularList class

Circular List Diagrams
- A circular list node is identical to a singly-linked list node.
- However, the circular list has a reference to its tail node instead of its head node.
- The tail node has a reference to the head node.
- This makes it possible to get to both ends of the list in constant time.

CircularList - State and Constructor
```java
public class CircularList implements List {
    protected int count;
    protected SinglyLinkedListElement tail;

    public CircularList() {
        // post: initializes the list to be empty,
        //       this.tail = null;
        this.count = 0;
    }
```

CircularList - peek() and tailPeek()
```java
public Object peek() {
    // pre: list is not empty
    // post: returns the first object in the list without
    // modifying the list
    return this.tail.next.value();
}

public Object tailPeek() {
    // pre: list is not empty
    // post: returns the last object in the list without
    // modifying the list
    return this.tail.value();
}
```
CircularList - Store Interface

/* Interface Store Methods */
public int size() { //post: returns the number of elements in the list.
    return this.size;
}
public boolean isEmpty() { //post: returns true if store is empty.
    return this.size == 0;
}
public void clear() { //post: clears the list so that it contains no elements.
    this.size = 0;
}

CircularList - Collection Interface 1

public boolean contains(Object anObject) { //pre: anObject is non-null
    //post: returns true if the collection contains the object
    SinglyLinkedListElement cursor;
    if (this.tail == null) { //empty list
        return false;
    } else {
        cursor = this.tail.next(); //partial traversal
        while (cursor != tail && !cursor.value.equals(anObject))
            cursor = cursor.next();
        return cursor.value.equals(anObject);
    }

CircularList - Collection Interface 2

public void add(Object anObject) { //pre: anObject is non-null
    //post: the object is added to the beginning of the list
    this.addToHead(anObject);
}
public Object remove(Object anObject) { //pre: anObject is non-null
    //post: removes object "equal" to anObject and returns
    //it, otherwise returns null
    // Similar to Singly-linked list removal.
    public Iterator elements() { //post: return an iterator for traversing the collection
        // Ignore this one until Iterators lecture
    }

CircularList - removeFromHead()

public Object removeFromHead() { //pre: list is not empty
    //post: removes and returns first object from the list
    SinglyLinkedListElement temp;
    temp = this.tail.next();
    if (this.tail == this.tail.next())
        this.tail = null;
    else
        this.tail.setNext(temp.next());
    this.count--; //Similar to Singly-linked list removal.
    return temp.value;
}

CircularList - addToHead()

public void addToHead(Object anObject) { //pre: anObject is non-null
    //post: the object is added to the beginning of the list
    SinglyLinkedListElement temp;
    temp = new SinglyLinkedListElement(anObject);
    if (this.tail == null) { //empty list
        this.tail = temp;
    } else {
        this.tail.setNext(temp);
        temp.setNext(this.tail.next());
        this.count++;
    }
}

CircularList - addToTail()

public void addToTail(Object anObject) { //pre: anObject is non-null
    //post: the object is added at the end of the list
    this.addToManyHead(anObject);
    this.tail = this.tail.next();
}
public Object removeFromTail() {
    // pre: list is not empty
    // post: the last object in the list is removed and returned
    SinglyLinkedListElement cursor;
    SinglyLinkedListElement temp;
    cursor = this.tail;
    if (cursor == this.tail) // one node list
        this.tail = null;
    else {
        cursor = cursor.next(); // move cursor to 2nd last node
        cursor.setNext(this.tail.next());
        this.tail = cursor;
        this.count--;
    }
    return temp.value();
}