Doubly Linked Lists

- Representation
- Space Analysis
- Creation and Insertion
- Traversal
- Deletion
public class DoublyLinkedList {
    protected Element head, tail;
    // . . .
    public class Element {
        Object data; Element next, previous;

        Element(Object obj, Element next, Element previous) {
            data = obj; this.next = next;
            this.previous = previous;
        }
        public Object getData() { return data; }
        public Element getNext() { return next; }
        public Element getPrevious() { return previous; }
        // . . .
    }
}

Representation

```
list
  └── head
      └── tail
         └── Element
            └── Element
```
Doubly Linked Lists : Space Analysis

- The space requirements of our representation of the doubly linked lists is as follows:

  \[ S(n) = \text{sizeof(DoublyLinkedList)} + n \times \text{sizeof(DoublyLinkedList.Element)} \]
  \[ = 2 \times \text{sizeof(DoublyLinkedList.Element ref)} + n \times [\text{sizeof(Object ref)} + 2 \times \text{sizeof(DoublyLinkedList.Element ref)}] \]
  \[ = (2n + 2) \times \text{sizeof(DoublyLinkedList.Element ref)} + n \times \text{sizeof(Object ref)} \]

<table>
<thead>
<tr>
<th>Required space</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sizeof(DoublyLinkedList)</td>
<td>The list reference has two fields: head (type: Element) and tail (type: Element) = 2 sizeof(DoublyLinkedList.Element ref)</td>
</tr>
<tr>
<td>n sizeof(DoublyLinkedList.Element)</td>
<td>The list has n elements of type Element. Each element has three fields-- previous (type Element), data (type Object), and next (type Element)</td>
</tr>
</tbody>
</table>
List Creation and Insertion

• An empty doubly linked list is created as follows:
  
  ```java
  DoublyLinkedList list = new DoublyLinkedList();
  ```

• Like singly link list, once Created, elements can be inserted into the list using either the `append` or `prepend` methods
  
  ```java
  for (int k = 0; k < 10; k++)
    list.append(new Int(k));
  ```

• Also if we have reference to a node (an element), we can use `insertAfter` or `InsertBefore` of the Element class.
public void append(Object obj){
    Element element = new Element(obj, null, tail);
    if(head == null)
        head = tail = element;
    else {
        tail.next = element;
        tail = element;
    }
}
public void prepend(Object obj) {
    Element element = new Element(obj, head, null);
    if (head == null)
        head = tail = element;
    else {
        head.previous = element;
        head = element;
    }
}

Complexity is O(1)
Insertion before an element

- Inserting before the current node (this) that is neither the first nor the last node:

  ```java
  Element element = new Element(obj, this, this.previous);
  this.previous.next = element;
  this.previous = element;
  ```

Complexity is $O(1)$
Traversal

For DoublyLinked list, traversal can be done in either direction. Forward, starting from head, or backward starting from tail.

```java
Element e = head;
while (e != null) {
    //do something
    e = e.next;
}
Element e = tail;
while (e != null) {
    //do something
    e = e.previous;
}
```

Example: Count the number of nodes in a linked list.

```java
public int countNodes(){
    int count = 0;
    Element e = head;
    while(e != null){
        count++;
        e = e.next;
    }
    return count;
}
```

Complexity is O(n)
Traversals

Example: The following computes the sum of the last n nodes:

```java
public int sumLastNnodes(int n){
    if(n <= 0)
        throw new IllegalArgumentException("Wrong: " + n);
    if(head == null)
        throw new ListEmptyException();

    int count = 0, sum = 0;
    Element e = tail;
    while(e != null && count < n){
        sum += ((Integer)e.data).intValue();
        count++;
        e = e.previous;
    }
    if(count < n)
        throw new IllegalArgumentException("No. of nodes < "+n);
    return sum;
}
```

Complexity is O(n)
Deletion

• To delete an element, we use either the `extract` method of `DoublyLinkedList` or that of the `Element` inner class.

```java
public void extract(Object obj){
    Element element = head;
    while((element != null) && (!element.data.equals(obj)))
        element = element.next;

    if(element == null)
        throw new IllegalArgumentException("item not found");
    if(element == head) {
        head = element.next;
        if(element.next != null)
            element.next.previous = null;
    }else{
        element.previous.next = element.next;
        if(element.next != null)
            element.next.previous = element.previous;
    }
    if(element == tail)
        tail = element.previous;
}
```

Complexity is O(n)
Exercises

• For the DoublyLinkedList class, Implement each of the following methods and state its complexity.
  – String toString()
  – Element find(Object obj)
  – void ExtractLast()
  – void ExtractFirst()
  – void ExtractLastN(int n)

• For the DoublyLinkedList.Element inner class, implement each of the following methods and state its complexity.
  – void insertBefore()
  – void insertAfter()
  – void extract()

• What are the methods of DoublyLinkedList and its Element inner class are more efficient than those of MyLinkedList class?