Last time: Single-linked lists

\[
\begin{array}{c}
\text{head} \\
\downarrow \text{content} \\
\rightarrow \text{next} \\
\end{array}
\]

Insertion may be \(O(1)\) or \(O(n)\) (first or last)

Finding an element (linear search)
// Inside the SingleLinked list class

public boolean find (Object c) {
if (head == null) return false;
    // isEmpty()

// Use cur as an Iterator (points to current place)
    Note cur = head;
    while (cur != null) {
        if (c.equals(cur.content)) return true;
        cur = cur.next; // go to next element
    }
    return false;
}
Linear processing in general:
Node cur = head;
while (cur != null) {
    // do something to content
    cur = cur.next;
}

Alternatively:
for (Node cur = head; cur != null; cur = cur.next) {
    // do something to cur, content
}
O(n)

Removing elements

```
head -> cur1 -> cur2 -> cur3
```

```
remove this element
```

Node cur = head;
head = cur.next

```
cur1.next = null;
cur1.content = null; // to allow GC to take
content that is no longer needed
```
public Object removeFirst() throws EmptyListException
{
    if (head == null) throw new EmptyListException();
    Node cur = head;
    head = head.next;
    Object o = cur.content;
    cur.content = null;
    cur.next = null;
    return o;
}
Double-Linked Lists

Node

prev  content  next

More flexibility; though O(1) same; more memory

public class Node {
    Object content;
    Node *next, *prev; // initialized to null
    // constructor ...
}

public class DoubleLinkedList {
    Node head, tail; // beginning & end of list
    // Insert in first position in a list

    1. head = tail = null
       (empty list)

    2. head = tail ≠ null
public void insertFirst(Object c) {
    Node m = new Node(c);
    // Link m at front of our list
    if (head
        m, next = head;  ①
    if (head != null) head.prev = m;
    head = m;
    // might need to adjust tail
    if (tail == null) // we had empty list before
        tail = m;
}
public void insertAfter (Object c, Object o) {
// Look for c in the list
Node curr = head;
while (curr != null) {
    if ((curr.content).equals (c)) break;
    curr = curr.next;
}
// Check how we terminated
if (curr == null) throw new Exception();
// we have content

Node newNode = new Node(0);
newNode.next = cur.next; // ①
cur.next = newNode; // ②
newNode.prev = cur;

if (cur.next != null) // if need be
    newNode.next.prev = newNode;

if (cur == tail) // // reference comparison
    tail = newNode; // not content comparison