

Intro to Programming II

Linked Lists

Chris Brooks

Department of Computer Science
University of San Francisco

18-2: Arrays

- Previously we talked about using arrays to store sequences of data.
- Advantages:
 - Everything is stored in sequential memory locations.
 - Fast lookup of elements.
- Disadvantages:
 - Hard to resize
 - Removing or adding an element in the middle is costly.

18-3: Linked Lists

- Linked lists have the opposite advantages and disadvantages
- Advantages:
 - Easy to insert and remove
 - Easy to resize
- Disadvantages:
 - Elements are not stored sequentially
 - Finding the nth element is slower.

18-4: Linked Lists

- The general idea:
- Each element of the list will “know” who the next element is.
- Let’s try this as a class.

18-5: List elements

- So how do we do this in Java?
- Our Element class needs to have two components:
 - The data that we want to store in the list.
 - A pointer to the next element in the list.

18-6: List elements

```
public class ListItem {
    public Object data;
    public ListItem next;

    public ListItem(Object d) {
        data = d;
        next = null;
    }
}
```

18-7: Arranging ListItems

- ListItems hook together like a chain.
- All we need to do is keep track of the beginning of the chain.
- No need to allocate everything ahead of time.

18-8: The LinkedList class

- The LinkedList will be responsible for hanging onto the 'head' of the list and providing methods for working with the list.
 - Insert()
 - InsertAt(index)
 - get(index)
 - remove(index)
 - find(object)

18-9: The LinkedList class

```
public class LinkedList {
    public ListItem head = null;
    public void insert(Object o) { ... }
    public void insertAt(Object o, int index) { ... }
    public Object get(int index) { ... }
    public Object remove(int index) { ... }
    public int find(Object o) { ... }
}
```

18-10: Adding

- So how do we add something to the front of a linked list?
 - Have the new thing point to the currently-first ListItem
 - Point 'head' to our new thing.

```
public void insert(Object o) {  
    ListItem l = new ListItem(o);  
    l.next = head;  
    head = l;  
}
```

18-11: Adding

- What about adding something into the middle of a list?
- If we want an item to go between current elements 5 and 6, then we need our new item to point to 6, and 5 to point to the new element.
- How do we code this?

18-12: Adding

```
public void insertAt(Object o, int index) {
    // first find the place to insert it.
    ListItem pointer = head;
    ListItem l = new ListItem(o);
    for (int i = 0; i < index - 1; i++) {
        pointer = pointer.next;
    }
    l.next = pointer;
    pointer = l;
}
```

18-13: Adding

- Are there any special cases we need to worry about?

18-14: Adding

- Are there any special cases we need to worry about?
 - What if the list is empty?
 - What if it only has one element?
 - What if we're adding at the end?

18-15: Exercise

- Code the `ListItem` and `LinkedList` classes as described above.
- Write a main method that creates a list and prompts the user for strings, which are always added to the front of the list.
- Add a method to the `LinkedList` class called `toString()`.
- This should walk the list and print out each of the strings in order.

18-16: Adding elements

- To add an element at the front, we point the new element's "next" pointer to whatever head is pointing to, then point head to point to the new element.

```
public void insert(Object o) {  
    ListItem l = new ListItem(o);  
    l.next = head;  
    head = l;  
}
```

18-17: Exercise

- So how do we iterate through a list and print out all the ListItems?

18-18: Exercise

- So how do we iterate through a list and create a string representing all the list contents? ListItems?

```
public String toString() {
    String result = '';
    ListItem current = head;
    while (current != null) {
        result += current.data;
        current = current.next;
    }
}
```

18-19: Adding elements

- What if we want to add an element in the middle of the list?
- First, we find where to put it, then we insert as if this was the front.

18-20: Adding elements by position

- The easiest way to add is by position.
- If we want an element to be in the n th position, we find the $n-1$ th element, and insert after that.

```
public void insertAt(String o, int index) {
    ListItem newItem = new ListItem(o);
    ListItem current = head;
    for (int i = 0; i < index; i++) {
        current = current.next;
    }
    newItem.next = current.next;
    current.next = newItem;
}
```

18-21: Adding elements by position

- Is there any way that this code could fail?

18-22: Adding elements by position

- Is there any way that this code could fail?
- What if *index* is larger than the number of elements in the list?

18-23: Adding elements by position

- Is there any way that this code could fail?
- What if *index* is larger than the number of elements in the list?
- Need to also make sure *current* is not null.

```
for (int i = 0; i < n && current != null; i++) {  
    ...  
}
```

18-24: Adding by order

- What if I want to add based on order?
 - Say we want to keep the list alphabetized?
- Look through until I find the right place.
- But: I need to be careful - I can't "back up" in the list.

18-25: Adding by order

```
public void insertAlpha(String name) {
    ListItem newItem = new ListItem(name);
    ListItem current = head;
    if (newItem.data.compareTo(current.data) <= 0) {
        insert(name);
    }
    while (current.next.data.compareTo(newItem.data) < 0) {
        current = current.next;
    }
}
```

18-26: Adding by order

- Wrinkles:
 - What about running off the end of the list?
 - What if the list is empty?
 - What if there's one element in the list?

18-27: Removing an element

- Removing is similar to finding by order.
- Conceptually, we need to link the item before the one to be removed to the item after the one to be removed.
- Remember: we can only go in one direction!

18-28: Removing an element

```
public String remove(String name)
    ListItem current = head;
    if (current.data.equals(name) ) {
        head = head.next;
        return current.data;
    } else {
        while ((current.next != null) &&
            !(current.next.data.equals(name) )) {
            current = current.next;
        }
        String rval = current.next.data;
        current.next = current.next.next ;
        return rval;
    }
```

18-29: Exercise

- Write a 'main' function that can insert people's names.
- Add a 'findBobs' method that will iterate over your list and find all people named Bob.
- Modify insertAlpha to insert names in reverse alphabetical order.
- Write a method called removeAt(int index) that removes the element at position index.