CSE 143 Java

Lists via Links

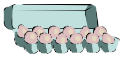
Reading: Ch. 23

Review: List Implementations

- The external interface is already defined
- Implementation goal: implement methods “efficiently”
- Array List approach: use an array with extra space internally
  - Iterating, indexing (get & set) is fast
    - Typically a one-liner
  - Adding at end is fast, except when we have to grow
  - Adding or removing in the middle is slow: requires sliding all later elements

A Different Way: Lists via Links

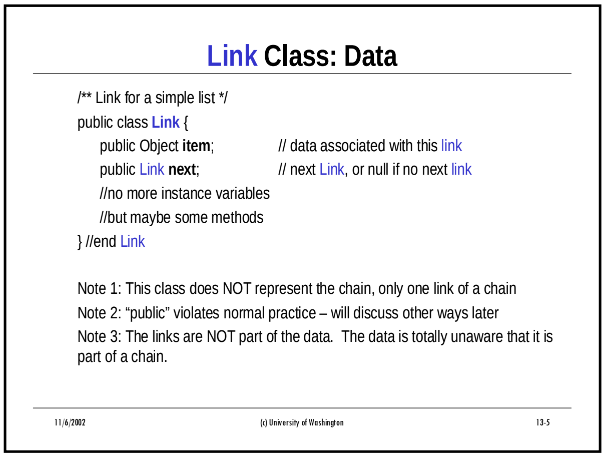
Instead of packing all elements together in an array,



create a *linked chain* of all the elements

Links and Lists

- For each element in the list, create a Link object
- The List object points to the *first* Link in the chain
- Each Link points to the *element* at that position, and also points to the *next* Link in the chain
- null marks the end of the list (chain)
- Each Link also points to the data being stored



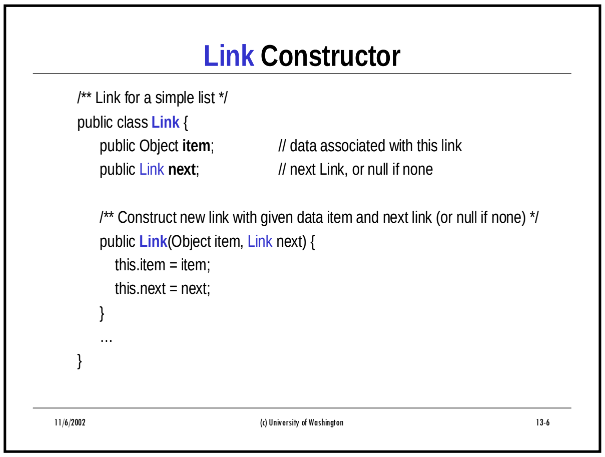
Link Class: Data

```
/** Link for a simple list */
public class Link {
    public Object item;          // data associated with this link
    public Link next;            // next Link, or null if no next link
    //no more instance variables
    //but maybe some methods
} //end Link
```

Note 1: This class does NOT represent the chain, only one link of a chain

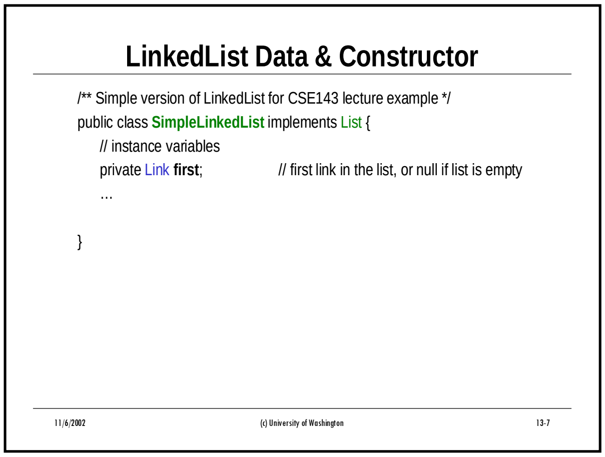
Note 2: “public” violates normal practice – will discuss other ways later

Note 3: The links are NOT part of the data. The data is totally unaware that it is part of a chain.



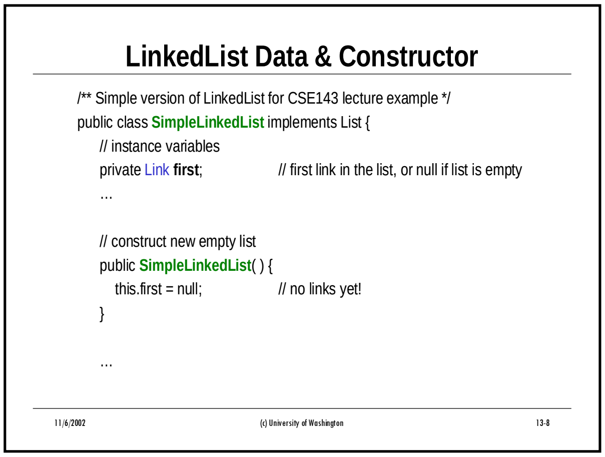
Link Constructor

```
/** Link for a simple list */
public class Link {
    public Object item;          // data associated with this link
    public Link next;            // next Link, or null if none

    /** Construct new link with given data item and next link (or null if none) */
    public Link(Object item, Link next) {
        this.item = item;
        this.next = next;
    }
    …
}
```



LinkedList Data & Constructor

```
/** Simple version of LinkedList for CSE143 lecture example */
public class SimpleLinkedList implements List {
    // instance variables
    private Link first;          // first link in the list, or null if list is empty
    …

}
```



LinkedList Data & Constructor

```
/** Simple version of LinkedList for CSE143 lecture example */
public class SimpleLinkedList implements List {
    // instance variables
    private Link first;          // first link in the list, or null if list is empty
    …

    // construct new empty list
    public SimpleLinkedList( ) {
        this.first = null;       // no links yet!
    }

    …
```

List Interface (review)

- Operations to implement:

```
int size( )
boolean isEmpty( )
boolean add(Object o)
boolean addAll(Collection other)
void clear( )
Object get(int pos)
boolean set(int pos, Object o)
int indexOf(Object o)
boolean contains(Object o)
Object remove(int pos)
boolean remove(Object o)
boolean add(int pos, Object o)
Iterator iterator( )
```

- **What don't we see anywhere here??**

Method add (First Try)

```
public boolean add(Object o) {
    // create new link and place at end of list:
    Link newLink = new Link(o, null);
    // find last link in existing chain: it's the one whose next link is null:
    Link p = this.first;
    while (p.next  !=  null) {
        p = p.next;
    }
    // found last link; now add the new link after it:
    p.next = newLink;
    return true;   // we changed the list => return true
}
```

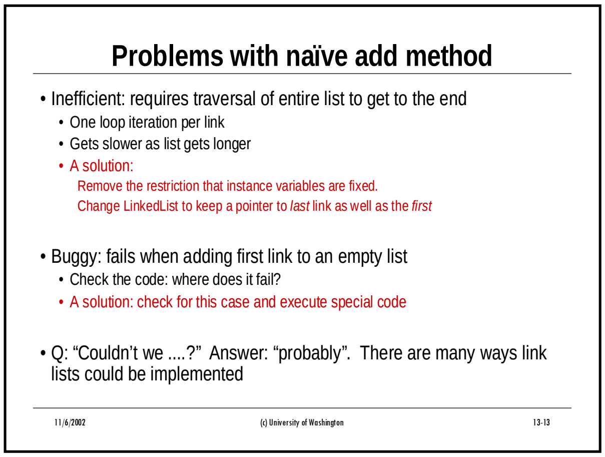
Draw the Official CSE143 Picture

- Client code:

```
LinkedList vertexes = new SimpleLinkedList();
Point2D p1 = new Point2D.Double(100.0, 50.0);
Point2D p2 = new Point2D.Double( 250, 310);
Point2D p3 = new Point2D.Double(90, 350.0);
vextexes.add(p1);
vertexes.add(p2);
vertexes.add(p3);
vertexes.add(p1);
```

Problems with naïve add method

- Inefficient: requires traversal of entire list to get to the end
  - One loop iteration per link
  - Gets slower as list gets longer
  - Solution??
- Buggy: fails when adding first link to an empty list
  - Check the code: where does it fail?
  - Solution??



Problems with naïve add method

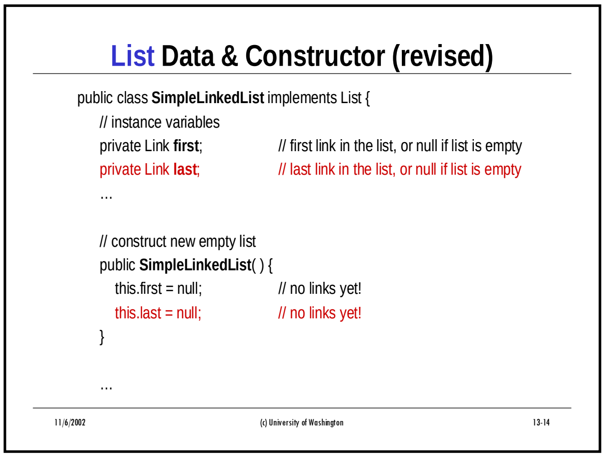
- Inefficient: requires traversal of entire list to get to the end
  - One loop iteration per link
  - Gets slower as list gets longer
  - A solution:

    Remove the restriction that instance variables are fixed.

    Change LinkedList to keep a pointer to *last* link as well as the *first*

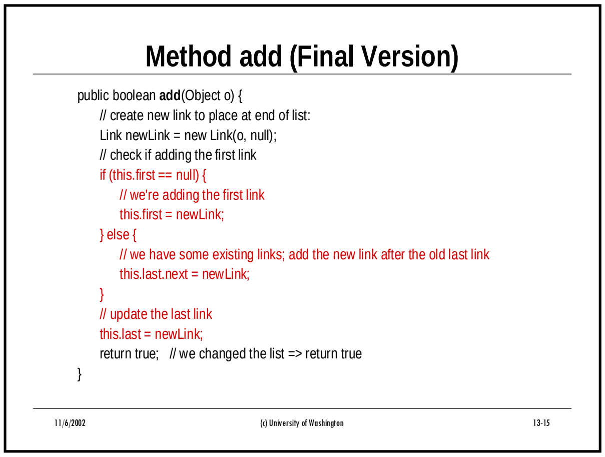
- Buggy: fails when adding first link to an empty list
  - Check the code: where does it fail?
  - A solution: check for this case and execute special code

- Q: “Couldn’t we ....?” Answer: “probably”. There are many ways link lists could be implemented



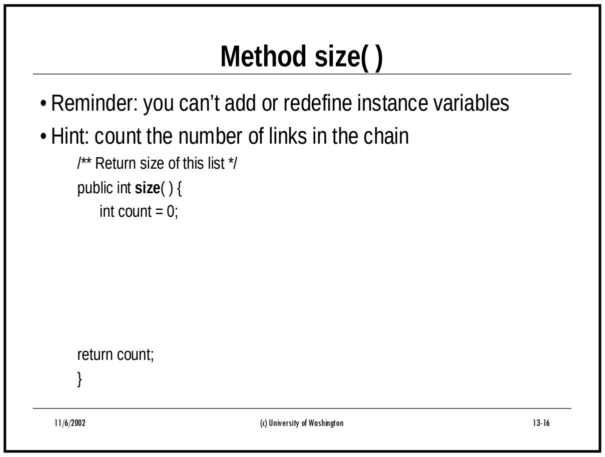
List Data & Constructor (revised)

```
public class SimpleLinkedList implements List {
    // instance variables
    private Link first;          // first link in the list, or null if list is empty
    private Link last;           // last link in the list, or null if list is empty
    …

    // construct new empty list
    public SimpleLinkedList( ) {
        this.first = null;       // no links yet!
        this.last = null;        // no links yet!
    }

    …
```



Method add (Final Version)

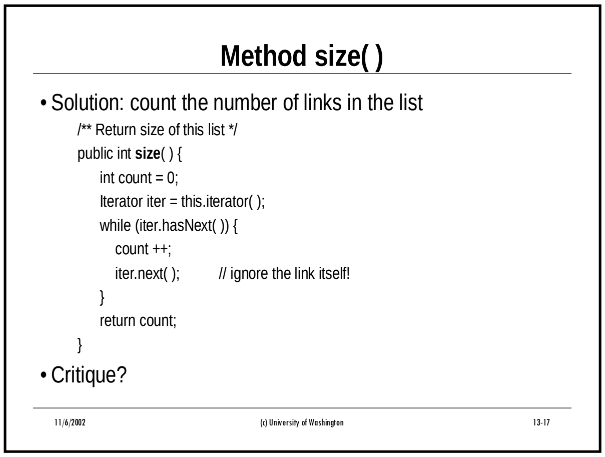
```
public boolean add(Object o) {
    // create new link to place at end of list:
    Link newLink = new Link(o, null);
    // check if adding the first link
    if (this.first == null) {
        // we're adding the first link
        this.first = newLink;
    } else {
        // we have some existing links; add the new link after the old last link
        this.last.next = newLink;
    }
    // update the last link
    this.last = newLink;
    return true;   // we changed the list => return true
}
```



Method size( )

- Reminder: you can’t add or redefine instance variables
- Hint: count the number of links in the chain

```
/** Return size of this list */
public int size( ) {
    int count = 0;




return count;
}
```

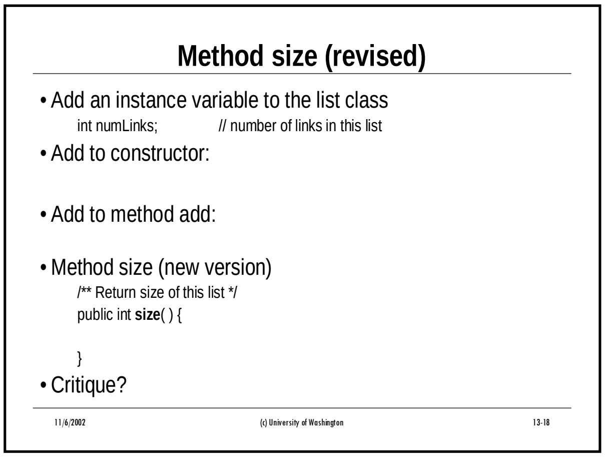


Method size( )

- Solution: count the number of links in the list

```
/** Return size of this list */
public int size( ) {
    int count = 0;
    Iterator iter = this.iterator( );
    while (iter.hasNext( )) {
        count ++;
        iter.next( );        // ignore the link itself!
    }
    return count;
}
```

- Critique?



Method size (revised)

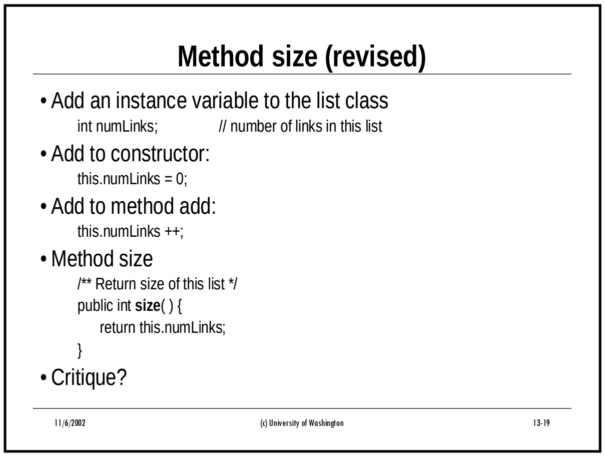
- Add an instance variable to the list class

```
int numLinks;        // number of links in this list
```

- Add to constructor:
- Add to method add:
- Method size (new version)

```
/** Return size of this list */
public int size( ) {

}
```

- Critique?



Method size (revised)

- Add an instance variable to the list class

```
int numLinks;        // number of links in this list
```

- Add to constructor:

```
this.numLinks = 0;
```

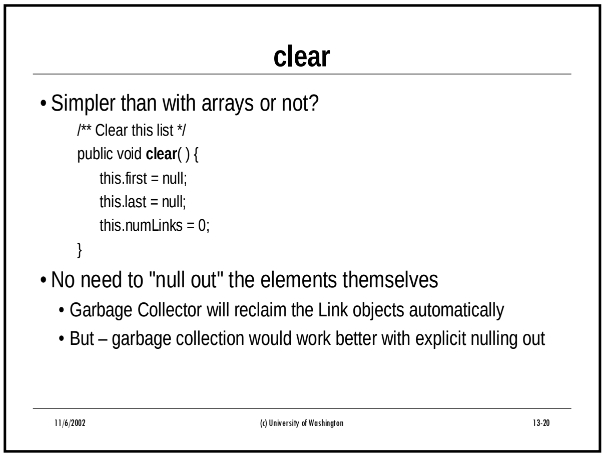
- Add to method add:

```
this.numLinks ++;
```

- Method size

```
/** Return size of this list */
public int size( ) {
    return this.numLinks;
}
```

- Critique?

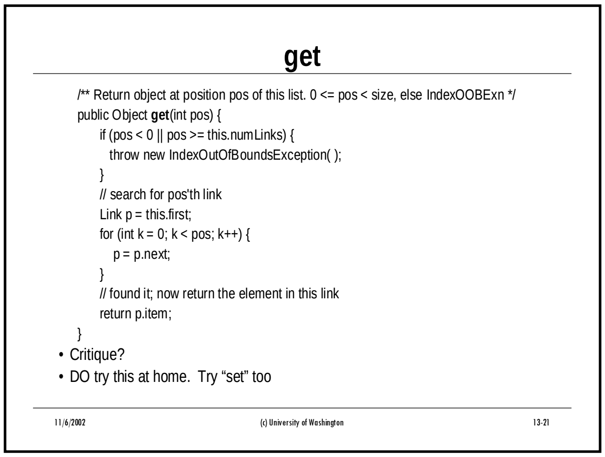


clear

- Simpler than with arrays or not?

```
/** Clear this list */
public void clear( ) {
    this.first = null;
    this.last = null;
    this.numLinks = 0;
}
```

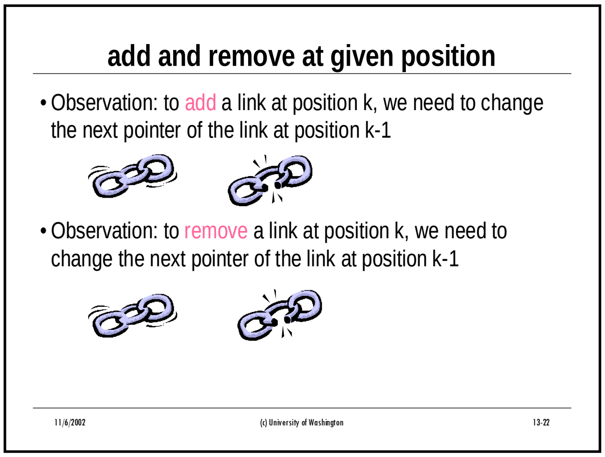
- No need to "null out" the elements themselves
  - Garbage Collector will reclaim the Link objects automatically
  - But – garbage collection would work better with explicit nulling out



get

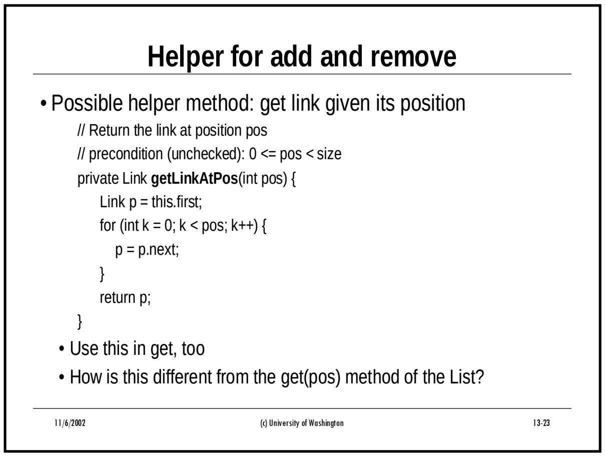
```
/** Return object at position pos of this list. 0 <= pos < size, else IndexOOBExn */
public Object get(int pos) {
      if (pos < 0 || pos >= this.numLinks) {
         throw new IndexOutOfBoundsException( );
      }
      // search for pos'th link
      Link p = this.first;
      for (int k = 0; k < pos; k++) {
          p = p.next;
      }
      // found it; now return the element in this link
      return p.item;
}
```

- Critique?
- DO try this at home. Try “set” too



add and remove at given position

- Observation: to add a link at position k, we need to change the next pointer of the link at position k-1
- Observation: to remove a link at position k, we need to change the next pointer of the link at position k-1

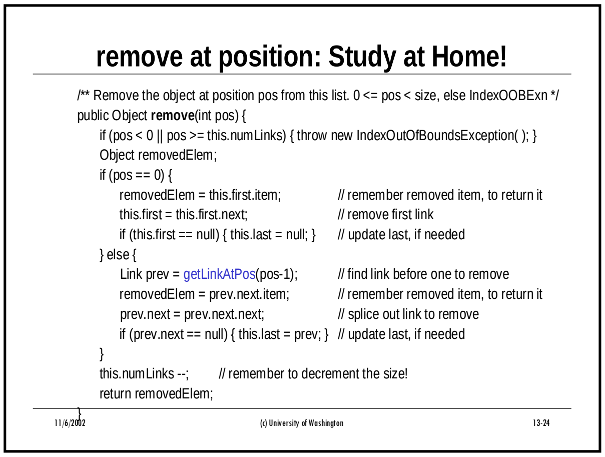


Helper for add and remove

- Possible helper method: get link given its position

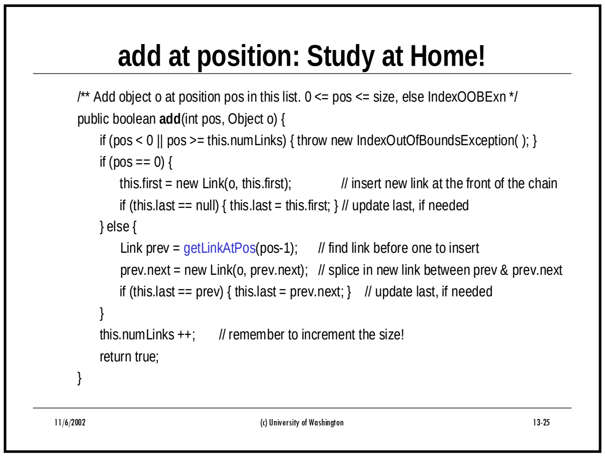
```
// Return the link at position pos
// precondition (unchecked): 0 <= pos < size
private Link getLinkAtPos(int pos) {
      Link p = this.first;
      for (int k = 0; k < pos; k++) {
          p = p.next;
      }
      return p;
}
```

- Use this in get, too
- How is this different from the get(pos) method of the List?



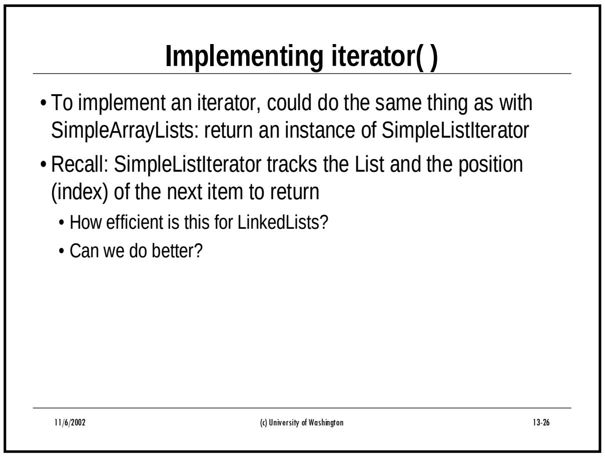
remove at position: Study at Home!

```
/** Remove the object at position pos from this list. 0 <= pos < size, else IndexOOBExn */
public Object remove(int pos) {
      if (pos < 0 || pos >= this.numLinks) { throw new IndexOutOfBoundsException( ); }
      Object removedElem;
      if (pos == 0) {
          removedElem = this.first.item;             // remember removed item, to return it
          this.first = this.first.next;              // remove first link
          if (this.first == null) { this.last = null; }   // update last, if needed
      } else {
          Link prev = getLinkAtPos(pos-1);           // find link before one to remove
          removedElem = prev.next.item;              // remember removed item, to return it
          prev.next = prev.next.next;                // splice out link to remove
          if (prev.next == null) { this.last = prev; }  // update last, if needed
      }
      this.numLinks --;      // remember to decrement the size!
      return removedElem;
}
```



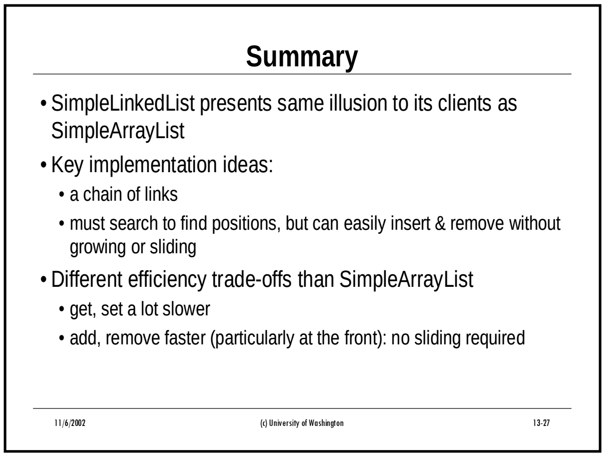
add at position: Study at Home!

```
/** Add object o at position pos in this list. 0 <= pos <= size, else IndexOOBExn */
public boolean add(int pos, Object o) {
    if (pos < 0 || pos >= this.numLinks) { throw new IndexOutOfBoundsException( ); }
    if (pos == 0) {
        this.first = new Link(o, this.first);          // insert new link at the front of the chain
        if (this.last == null) { this.last = this.first; } // update last, if needed
    } else {
        Link prev = getLinkAtPos(pos-1);     // find link before one to insert
        prev.next = new Link(o, prev.next);   // splice in new link between prev & prev.next
        if (this.last == prev) { this.last = prev.next; }    // update last, if needed
    }
    this.numLinks ++;      // remember to increment the size!
    return true;
}
```



Implementing iterator( )

- To implement an iterator, could do the same thing as with SimpleArrayLists: return an instance of SimpleListIterator
- Recall: SimpleListIterator tracks the List and the position (index) of the next item to return
  - How efficient is this for LinkedLists?
  - Can we do better?



Summary

- SimpleLinkedList presents same illusion to its clients as SimpleArrayList
- Key implementation ideas:
  - a chain of links
  - must search to find positions, but can easily insert & remove without growing or sliding
- Different efficiency trade-offs than SimpleArrayList
  - get, set a lot slower
  - add, remove faster (particularly at the front): no sliding required