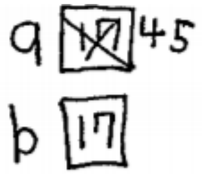
CSSE 220

Linked List Implementation
and
Project Preparation

Import *LinkedListSimple* project from repo

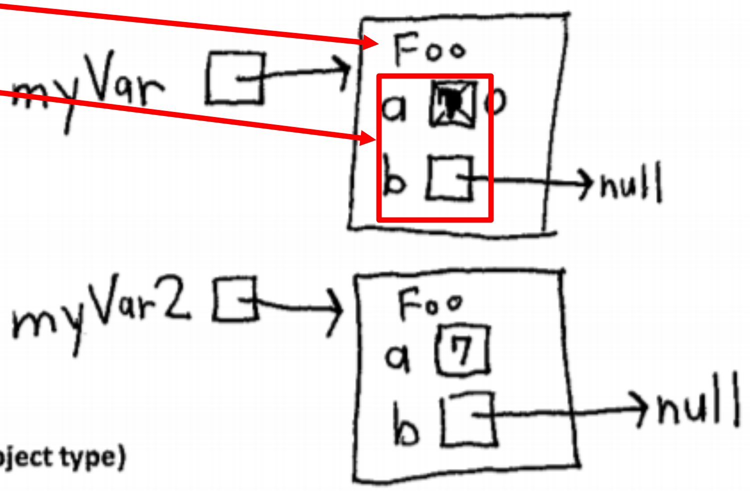
```
int a = 17;
int b = a;
a = 45;
```



A new for a class

A new for a class creates a new instance of a class. You should make a new rectangle for that class, label it with that class's name, and fill in all the fields for the class (according to the constructor of the class). Note that fields follow all the same rules as normal variables. Make the variable being assigned point to that array. *Note that without a "new", no new instances of a class (rectangles) can be created.*

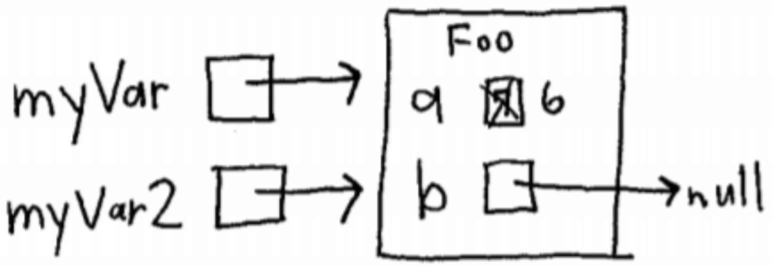
```
class Foo {
   public int a;
   public OtherClass b;
   public Point() {
      a = 7;
      b = null;
   }
}
Foo myVar = new Foo();
Foo myVar2 = new Foo();
myVar.a = 0;
```



An assignment of a non-primitive type (object type)

If you see an assignment of a non-primitive type, that copies the reference (i.e. that makes the variables point to the same object). So the arrow of the assigned object points to whatever the original object pointed to.

```
Foo myVar = new Foo();
Foo myVar2 = myVar;
myVar.a = 6;
```



Quiz - Today Front Page Only
Back Page On Friday

- Get into pairs
- Look at/run the code in LinkedList.java main
- Draw a box-and-pointer diagram of what's happening in the main code.
- To figure it out, you'll have to look at the LinkedList constructor and addAtBeginning.
- If you've forgotten how to do box-and-pointer diagrams, checkout the handout on Day 5 of the schedule

Motivation For Linked Structures

- Implementing container components - minimum
  - An *add* method
  - A *remove* method
  - A *get-size* method

- Two types of container components:

1. Bounded components – array based
2. Unbounded components – linked structures

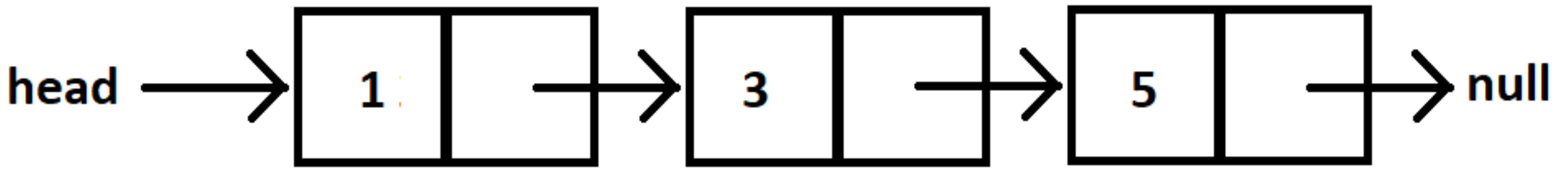
Solve Remaining Methods in LinkedListSimple

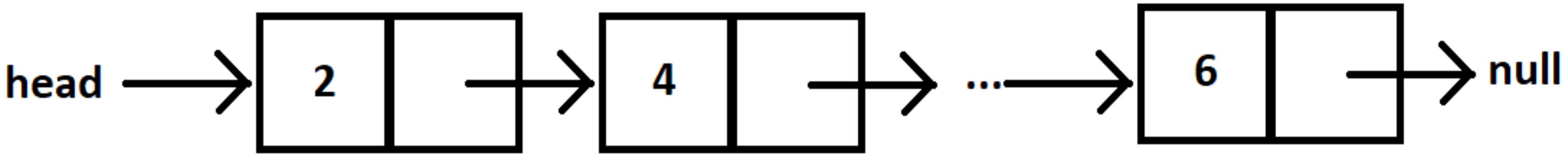
- Look at toString to get an idea of how to do size, then go from there
- They are in approximate difficulty order
- Get help if you get stuck!

- Hold on to your quiz today, we will finish it next class period.

Shorthand Notation

- Using pictures will be extremely helpful
- Can use System.out.println(this) to see what the current list looks like (does it match diagram?)





Loops in Arrays vs. LinkedLists

```
int[]  nums = ……


for (int i=0; i< nums.length; ++) {
       //do stuff with
       //arbitrary element nums[i]
}

Equivalent in while loop

int i=0;
while ( ?  ) {
       //do stuff with
       //arbitrary element nums[i]
       ?
}
```

```
LinkedList list = ……


//Another Day!


Node current = this.head;
while ( ?  ) {
       //do stuff with
       //arbitrary element
       ?
}
```

Loops in Arrays vs. LinkedLists

```
int[]  nums = ……

for (int i=0; i< nums.length; ++) {
       //do stuff with
       //arbitrary element nums[i]
}

Equivalent in while loop

int i=0;
while ( i < nums.length  ) {
       //do stuff with
       //arbitrary element nums[i]
       i++;
}
```

```
LinkedList list = ……

//Another Day!
```

Use a ‘for’ because we know exact length of array at outset
This is known as a *definite loop*

```
Node current = this.head;
while ( ?  ) {
       //do stuff with
       //arbitrary element
       ?
}
```

Loops in Arrays vs. LinkedLists

```
int[]  nums = ……


for (int i=0; i< nums.length; ++) {
       //do stuff with
       //arbitrary element nums[i]
}

Equivalent in while loop

int i=0;
while ( i < nums.length  ) {
       //do stuff with
       //arbitrary element nums[i]
       i++;
}
```

```
LinkedList list = ……


//Another Day!







Node current = this.head;
while ( ?  ) {
       //do stuff with
       //arbitrary element
       current = current.next;
}
```

Loops in Arrays vs. LinkedLists

```
int[]  nums = ……


for (int i=0; i< nums.length; ++) {
       //do stuff with
       //arbitrary element nums[i]
}

Equivalent in while loop

int i=0;
while ( i < nums.length  ) {
       //do stuff with
       //arbitrary element nums[i]
       i++;
}
```

```
LinkedList list = ……


//Another Day!
```

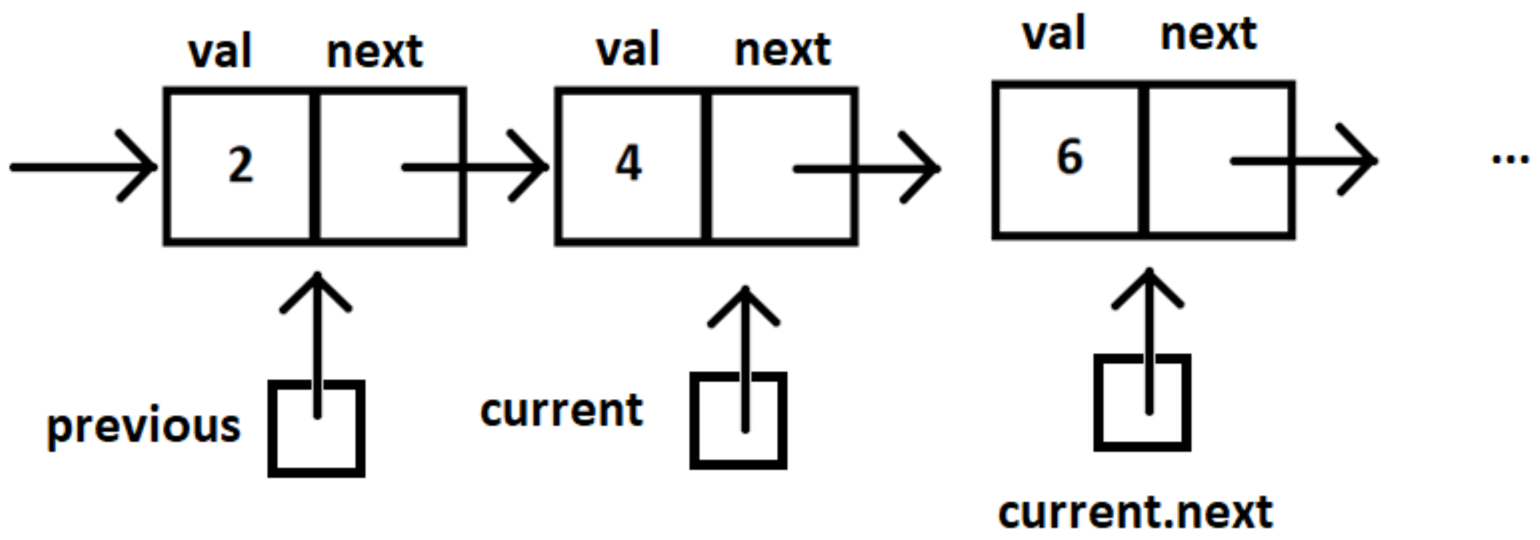
Use a ‘while’ because we do not know exact length of linked list at outset
This is known as an *indefinite loop*

```
Node current = this.head;
while ( current != null  ) {
       //do stuff with
       //arbitrary element
       current = current.next;
}
```

Solve the Other Methods in LinkedListSimple

- Look at toString to get an idea of how to do size, then go from there
- They are in approximate difficulty order
- Get help if you get stuck!
  - size()
  - add…
  - remove…

Shorthand Notation



Homework

- SinglyLinkedList
  - Requires you to implement a SinglyLinkedList
  - Additional algorithm questions which make use of the SinglyLinkedList
  - Will give time in next class to work on it

Software Engineering Techniques

- Pair programming
  - Project can be coordinated well with this
- Version Control
  - How to avoid merge conflicts when using git

What Is Pair Programming?

- Two programmers work side-by-side at a computer, continuously collaborating on the same design, algorithm, code, and/or test
- Enable the pair to produce higher quality code than that produced by the sum of their individual efforts

Pair Programming

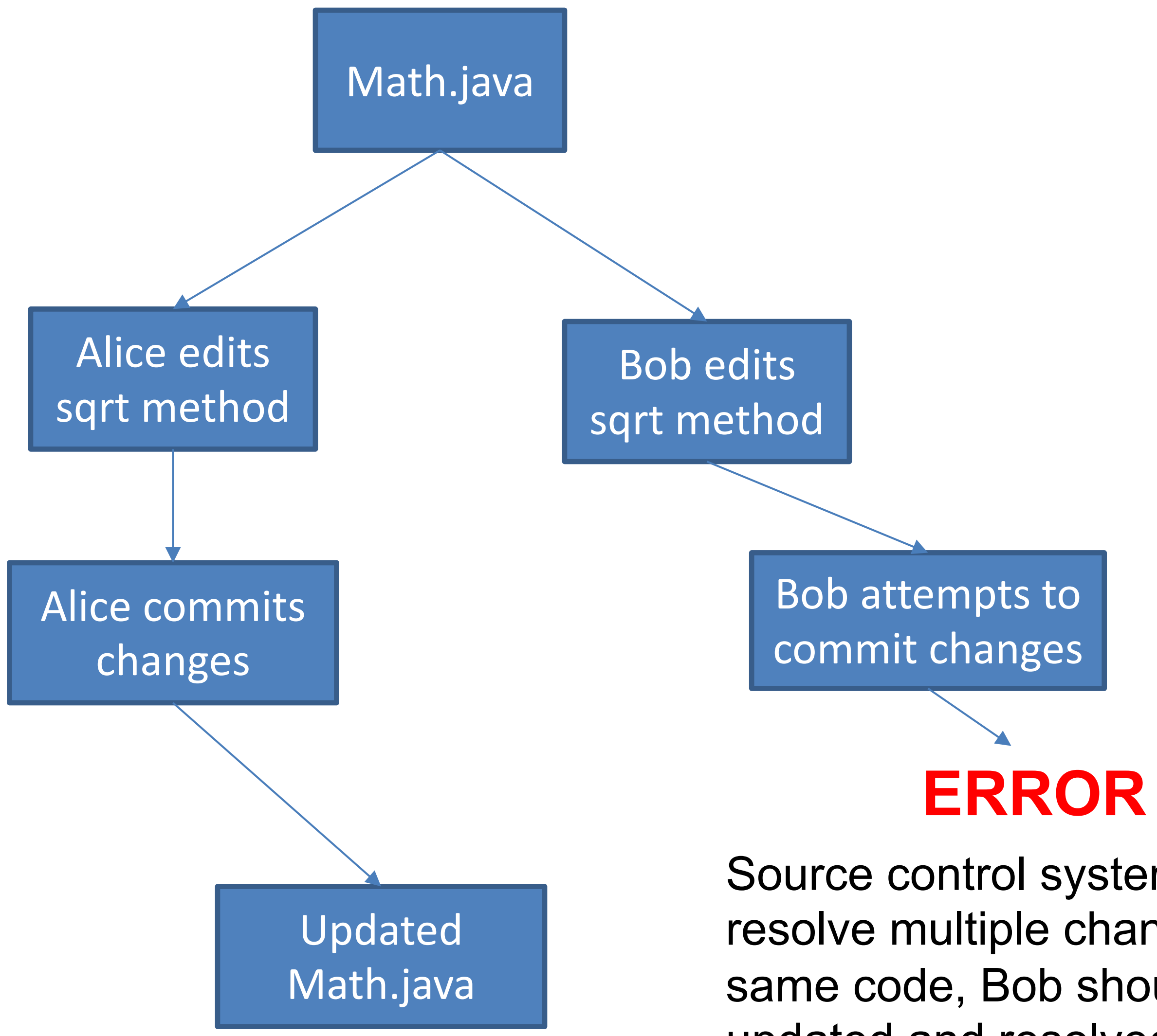
- Working in pairs on a single computer
  - The *driver*, uses the keyboard, talks/thinks out-loud
  - The *navigator*, watches, thinks, comments, and takes notes
  - Person who really understands should start by navigating ☺
- For hard (or new) problems, this technique
  - Reduces number of errors
  - Saves time in the long run

Pair programming video

- https://www.youtube.com/watch?v=rG_U12uqRhE

SOFTWARE VERSIONS

When Two+ People Edit the Same Code



Source control system cannot resolve multiple changes on the same code, Bob should have updated and resolved conflicts before committing.

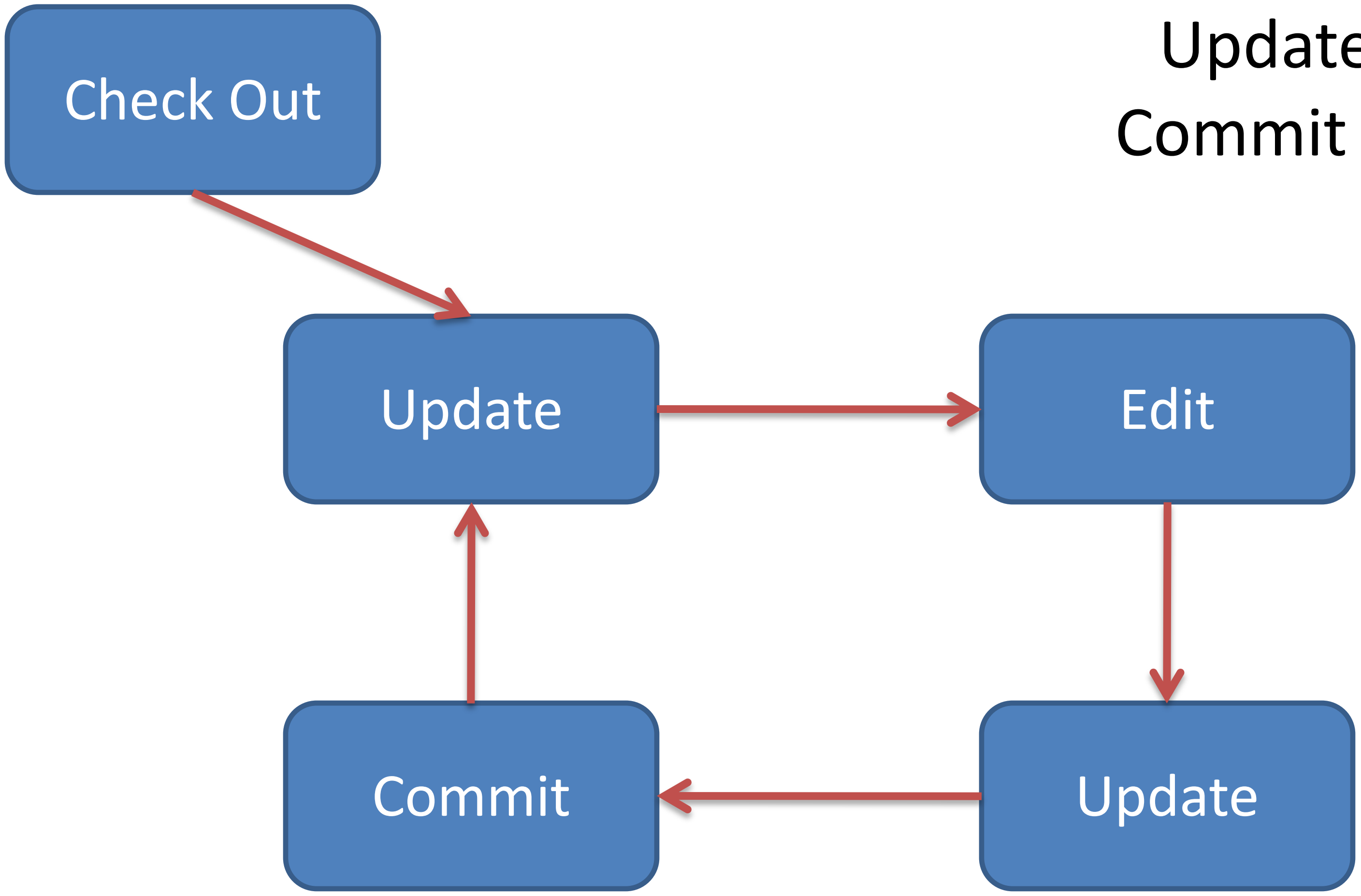
Team Version Control

- **Version control tracks multiple versions**
  - Enables old versions to be recovered
  - Allows multiple versions to exist simultaneously

- **Always**:
  - **Update before** working
  - **Update again** before committing
  - **Commit often** and with good messages

- **Communicate** with teammates so you don’t edit the same code simultaneously
  - Pair programming ameliorates this issue ☺

Team Version Control



Update and
Commit often!

What if I get a conflict on update?

- If you did an update and now have File.java, File.java.mine, File.java.rN, and File.java.rM (where N and M are integers):
  - YOU HAVE A CONFLICT!
- Eclipse provides tools for resolving conflicts
- Follow the steps in this link to resolve a conflict:
  - http://www.rose-hulman.edu/class/csse/csse221/current/Resources/ResolvingSubversionConflicts.htm

TEAM PROJECT WORK TIME

- Move into your groups if not already
- Review comments from Milestone 0 feedback
- Be prepared to ask question of the grader
- You will have ~5 minutes, so use it well

WEDNESDAYS MATERIAL

CSSE 220

DATA STRUCTURES + BIG-O NOTATION

Understanding the engineering trade-offs when storing data

Import *LinkedListSimple* project from repo

Import *SinglyLinkedList* homework from repo

Data Structures

- Efficient ways to store data **based on how we will be using it**

- The main theme for the rest of the course

- So far we've seen `ArrayLists`
  - Fast addition **to end of list**
  - Fast access to any existing position
  - Slow *inserts into* and *deletes from* middle of list

Big-O Notation

- Describes the limiting behavior
  - How slow it can possibly run?
  - Describes the <u>worst case</u>
- Used for Classifying Algorithm Efficiency
- “O” stands for “Order”
  - O(n) → said as “Order n”
  - O(n^2) → said as “Order n-squared”

Big-O Notation (continued)

1. Don't Care About Constant Coefficients
   - f(n) = $\cancelto{1}{2}n$ → f(n) is O(n)
   - To get Big-O for f(n) change constant coefficients to 1

Q2

Big-O Notation (continued)

1. Don't Care About Constant Coefficients
   - $f(n) = \overset{1}{\cancel{2}}n \rightarrow f(n)$ is $O(n)$
2. Don't Care About Lower Order Terms
   - $f(n) = \overset{1}{\cancel{6}}n^2 + \xcancel{7n^1} + \xcancel{3n^0} \rightarrow f(n)$ is $O(n^2)$
   - Eliminate lower order terms

Q2

Big-O Notation (continued)

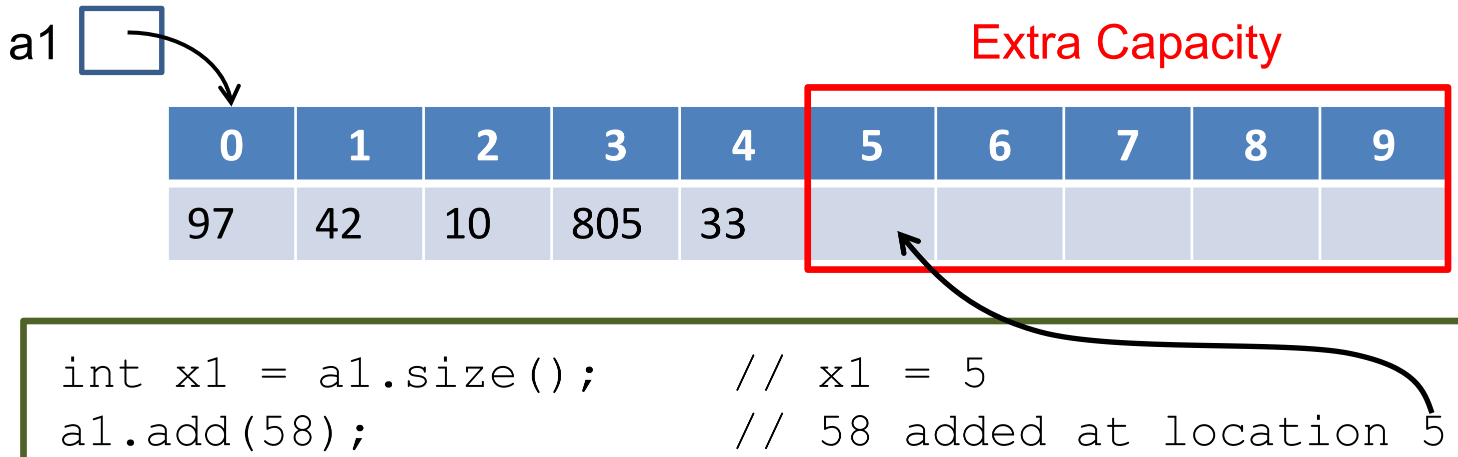
1. Don't Care About Constant Coefficients
   - $f(n) = \overset{1}{\cancel{2}}n \rightarrow f(n)$ is $O(n)$
2. Don't Care About Lower Order Terms
   - $f(n) = \overset{1}{\cancel{6}}n^2 + \cancel{7n} + \cancel{3n^0} \rightarrow f(n)$ is $O(n^2)$

- If $f(n)$ is constant, we say $O(1) \rightarrow$ "Order 1"
  - For example, $f(n) = 48 \rightarrow f(n)$ is $O(1)$
  - But really when $f(n) = 48$
    then $f(n) = \overset{1}{\cancel{48}}n^0 \rightarrow f(n)$ is $O(n^0)$ which is $O(1)$

Q2

ArrayList Performance (Revisited)

- Fast addition to end of list:
  - Fast access to any existing position – O(1) (like array)
  - Keep extra *capacity* for list growth
    - ArrayList<Integer> a1;
    - Fast access includes items in capacity not yet filled – O(1)



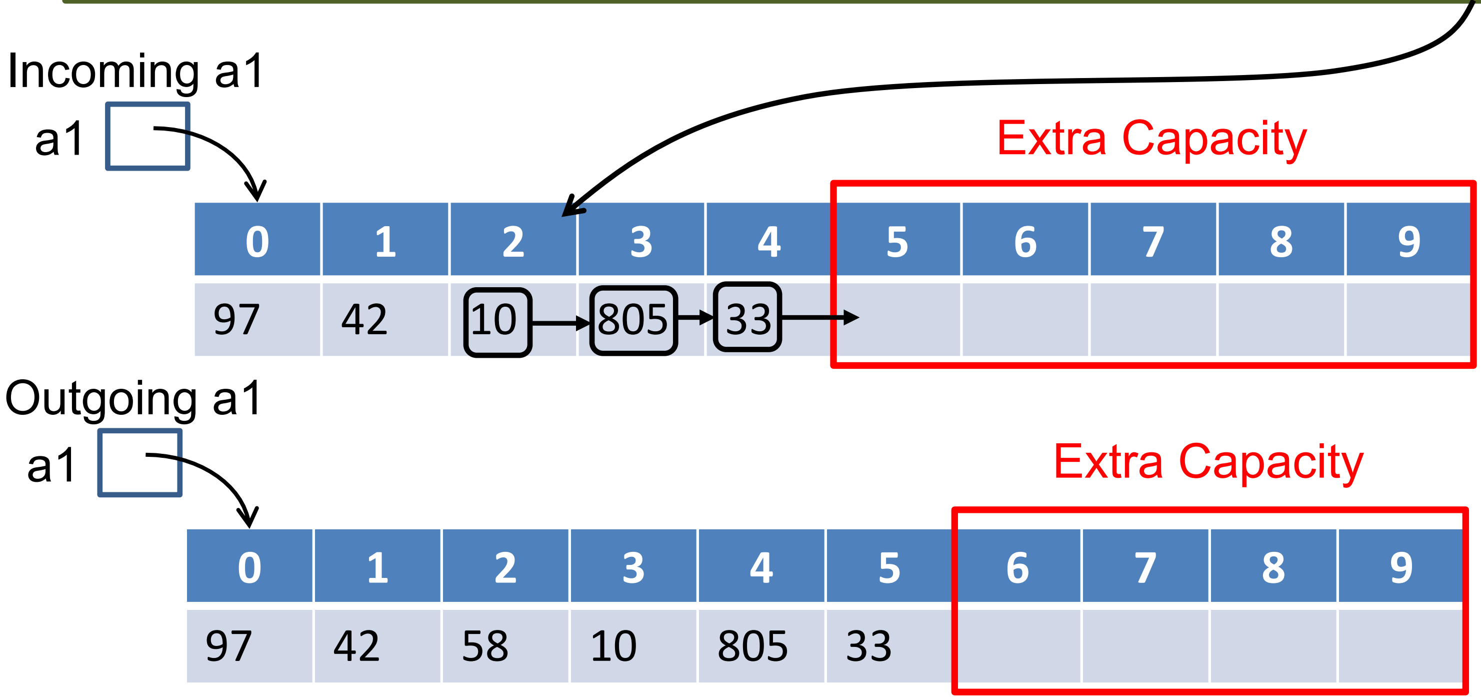
```
int x1 = a1.size();        // x1 = 5
a1.add(58);                // 58 added at location 5
x1 = a1.size();            // now x1 = 6
```

Q3

ArrayList Performance (Revisited)

- Slow *inserts into* middle of list
  - For insert, shift all items right to accommodate -O(n)

```
int x1 = a1.size();      // x1 = 5
a1.add(2,58);            // 58 added at location 2
```

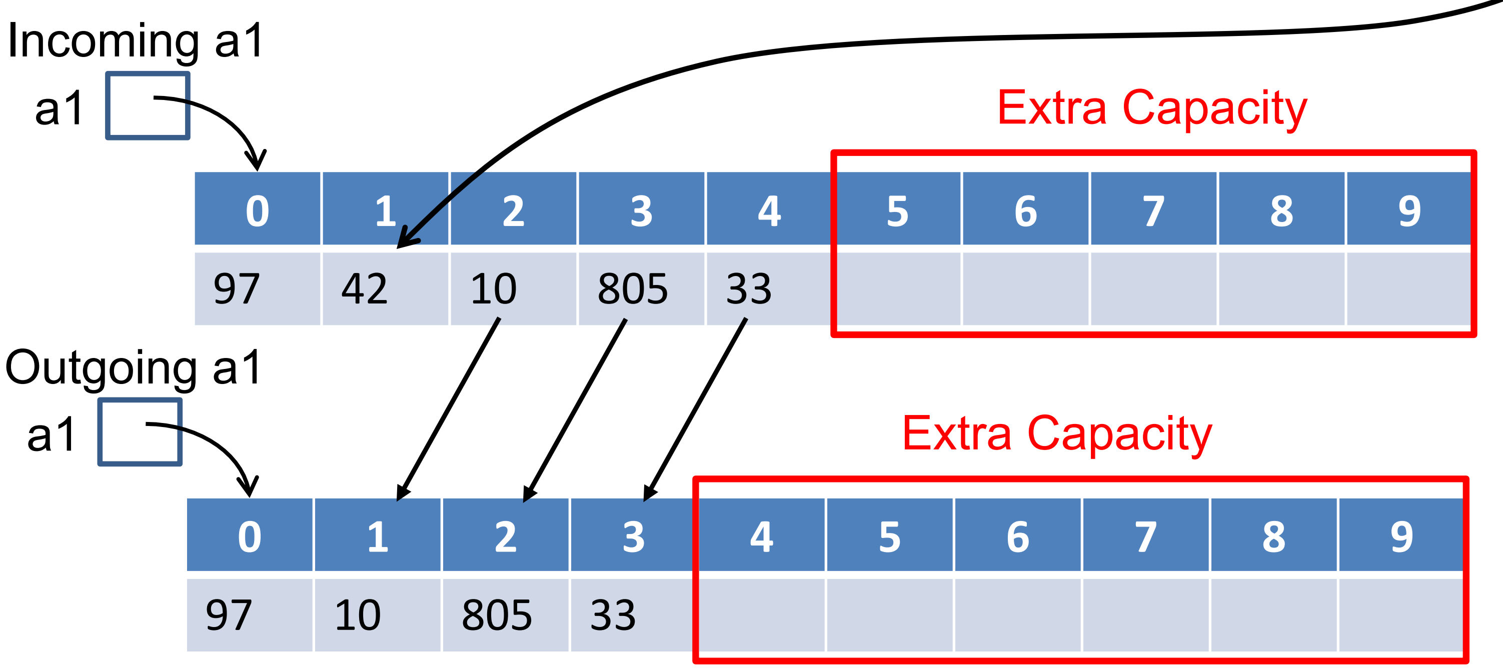


Q3

ArrayList Performance (Revisited)

- Slow *deletes from* middle of list
  - For remove, shift all items to fill hole created -O(n)

```
int x1 = a1.remove(1);  // remove at location 1
                        // x1 gets 42
```



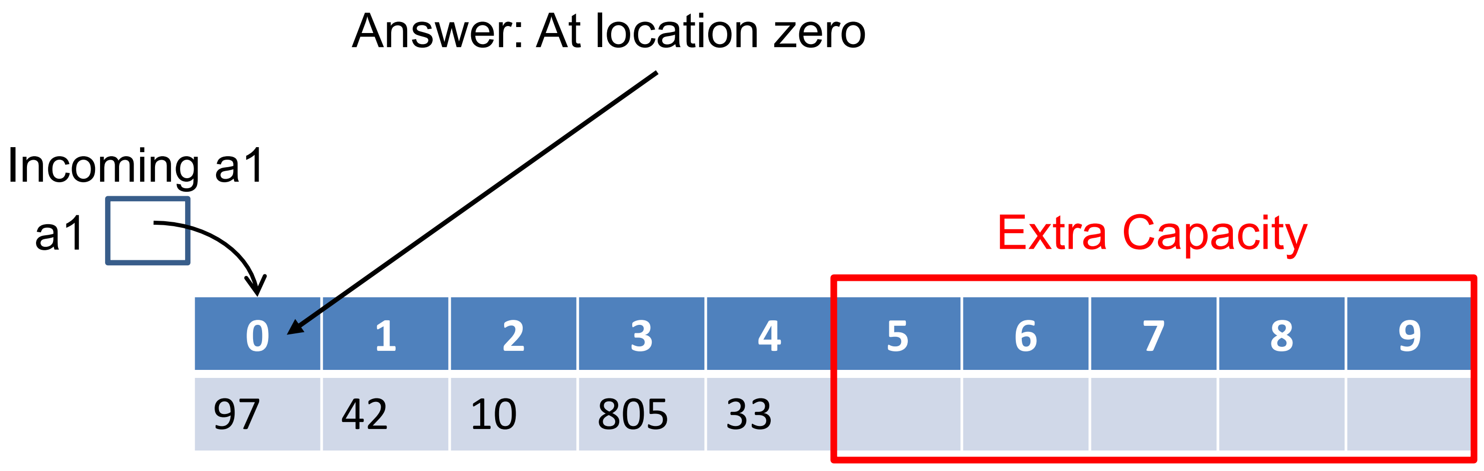
Q3

ArrayList Performance (Revisited)

- Where in ArrayList is worst case for:
  - Inserting into?
  - And deleting from?

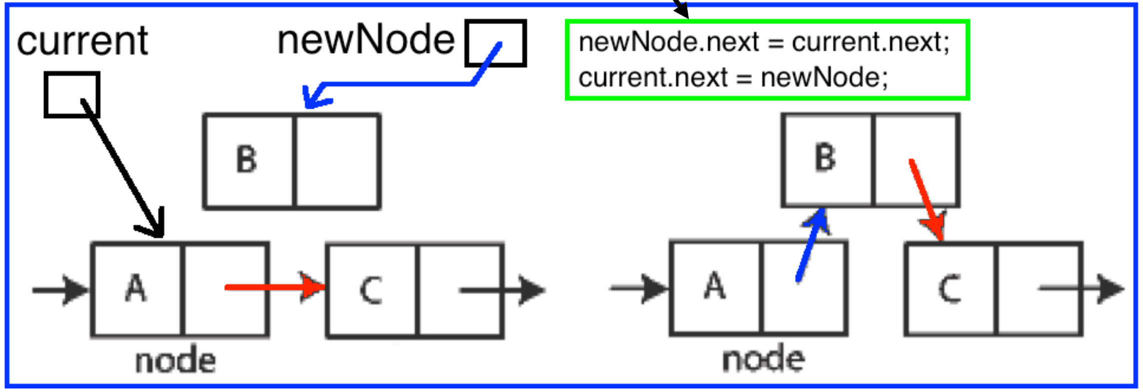
ArrayList Performance (Revisited)

- Where in ArrayList is worst case for:
  - Inserting into?
  - And deleting from?



Another List Data Structure

- What if we have to add/remove data from a list frequently?
- `LinkedLists` support this:
  - Fast insertion and removal of elements
    - Once we have the *current* pointer
    - Execute 2 lines of code

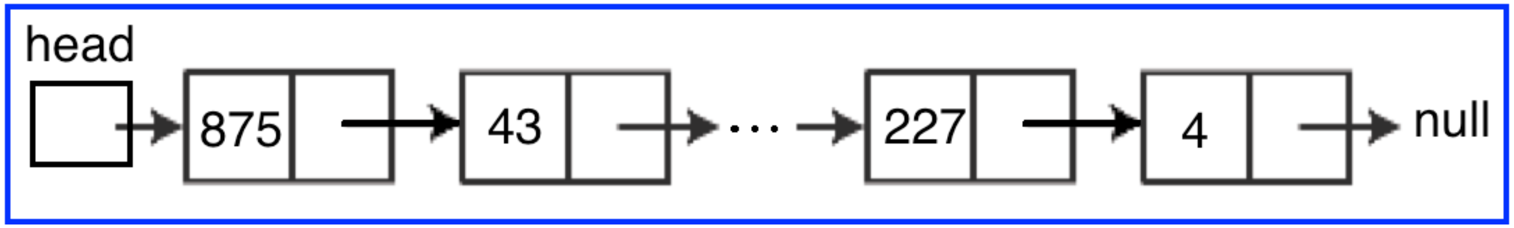


```
newNode.next = current.next;
current.next = newNode;
```

Q4-5

Another List Data Structure

- `LinkedLists` support this:
  - Slow access to arbitrary elements
  - Why?
  - Because we always have to start at ‘head’ and traverse the List to find the Node containing what we are looking for



Q4-5

`LinkedList<E>` Methods

- **`void addFirst(E element)`**
- **`void addLast(E element)`**
- **`E getFirst()`**
- **`E getLast()`**
- **`E removeFirst()`**
- **`E removeLast()`**

Complete Quiz

- Turn in quiz today

Homework

- SinglyLinkedList
  - Requires you to implement a SinglyLinkedList
  - Additional algorithm questions which make use of the SinglyLinkedList
  - Will give you remaining class time to work on it
  - If you complete it, work on the project!