

# CSSE 220

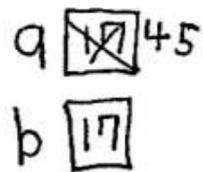
Linked List Implementation  
and  
Project Preparation

Checkout *LinkedListSimple* project from SVN

# Quiz

- 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

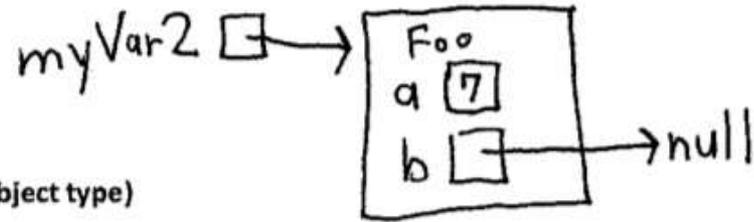
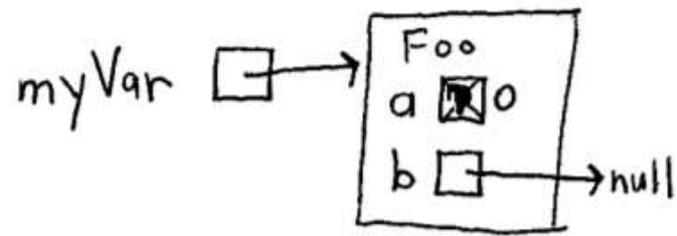
```
int a = 17;
int b = a;
a = 45;
```



**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.

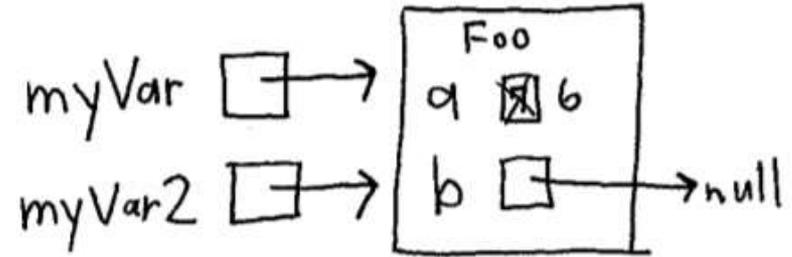
```
class Foo {
    public int a;
    public OtherClass b;
    public Point() {
        a = 7;
        b = null;
    }
}
Foo myVar = new Foo();
Foo myVar2 = new Foo();
myVar.a = 0;
```



**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;
```



# Solve the Other Problems 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; ++i) {  
    //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!
```

```
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 = .....
```

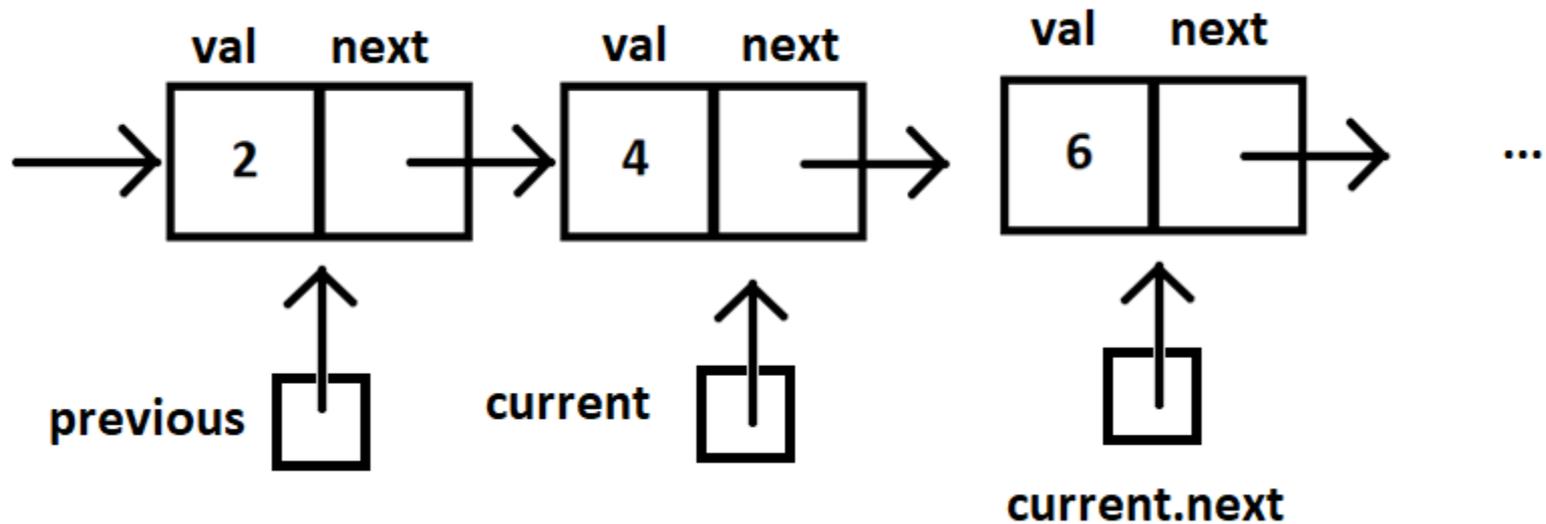
```
//Another Day!
```

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

# Solve the Other Problems 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 in SVN

# 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

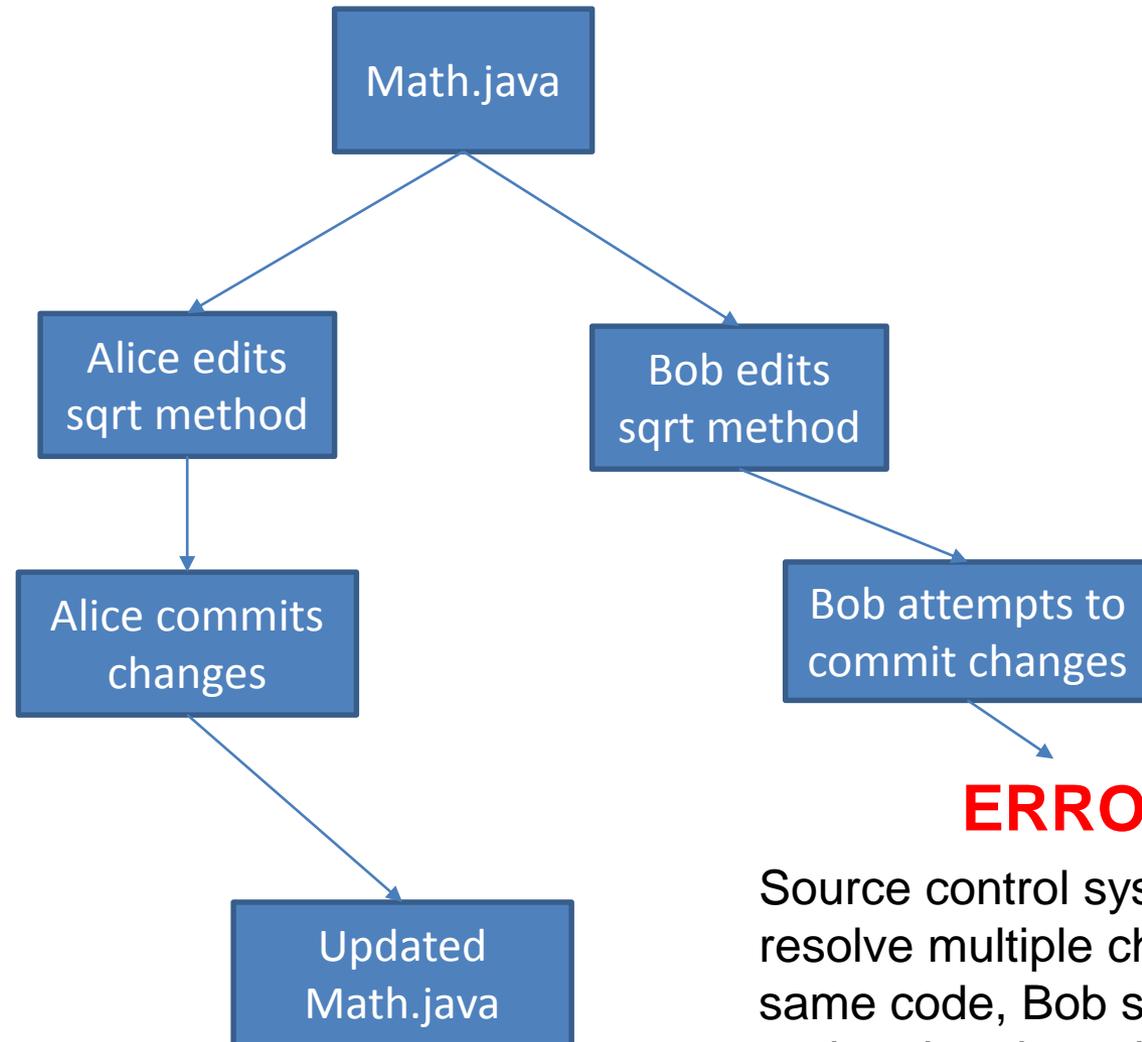
- Working in pairs on a single computer
  - The *driver*, uses the keyboard, talks/thinks out-loud
  - The *navigator*, watches, thinks, comments, and takes notes
  - Person who really understands should start by navigating 😊
- For hard (or new) problems, this technique
  - Reduces number of errors
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

# Pair programming video

- [https://www.youtube.com/watch?v=rG\\_U12uqRhE](https://www.youtube.com/watch?v=rG_U12uqRhE)

# **SOFTWARE VERSIONS**

# When Two+ People Edit the Same Code



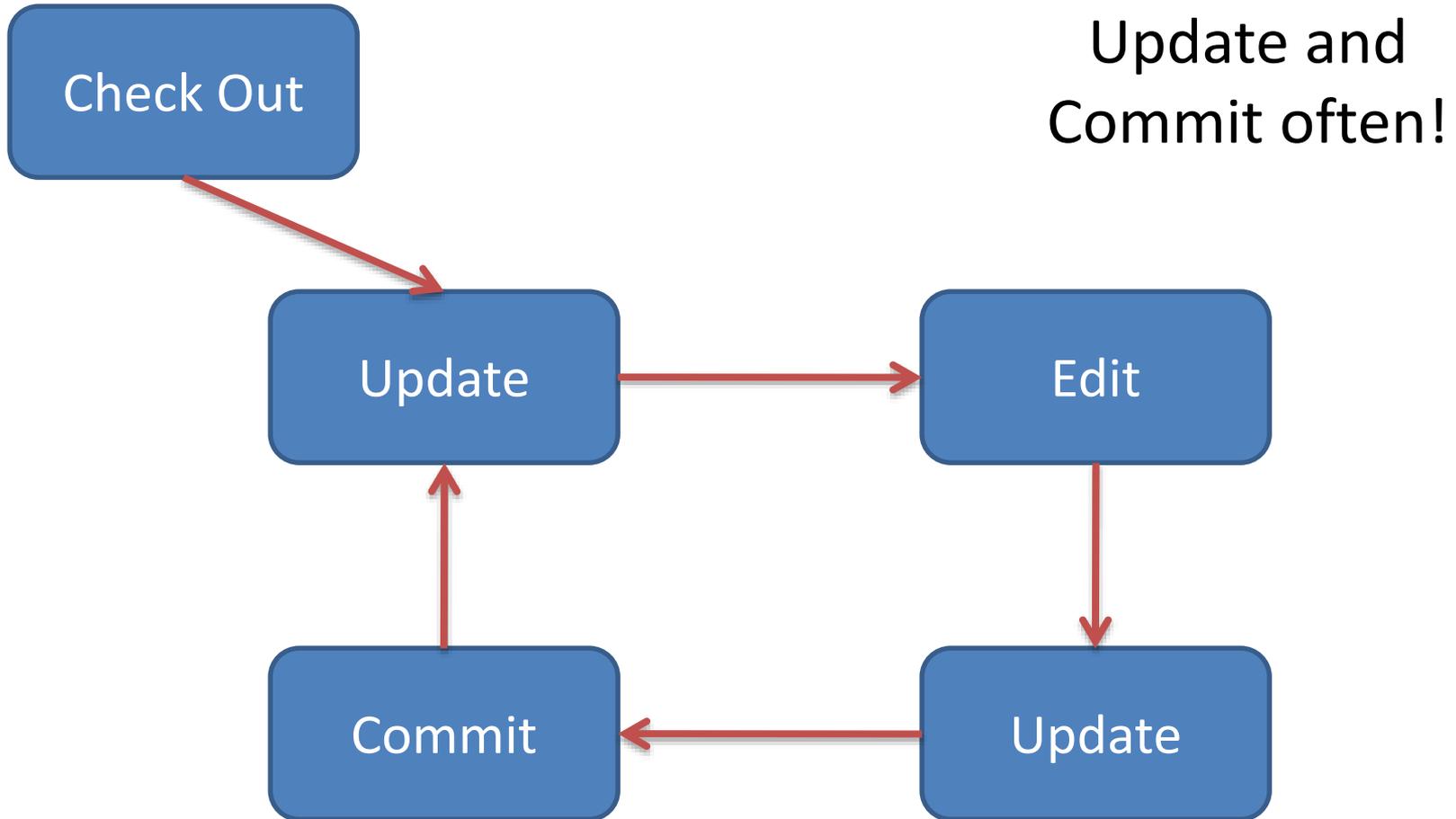
**ERROR**

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



# 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

Checkout *LinkedListSimple* project from SVN

Checkout *SinglyLinkedList* homework from SVN

# Data Structures

- Efficient ways to store data **based on how we'll use 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 to and deletes from middle of list

# Big-O Notation

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

# Big-O Notation (continued)

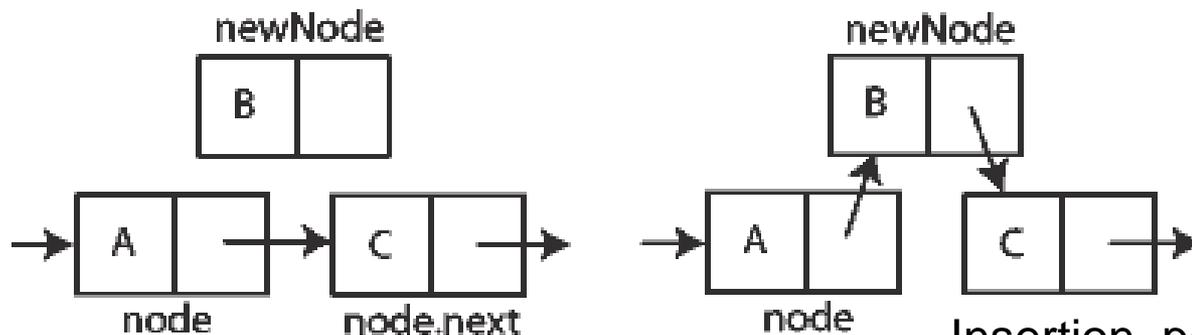
- Don't Care About Constants
  - $O(2n + 7) \rightarrow O(n)$
- Don't Care About Smaller Powers
  - $O(6n^2 + 7n) \rightarrow O(n^2)$
  - Algorithm grows asymptotically no faster than  $n^2$
- If constant value, we say  $O(1) \rightarrow$  “Order 1”
  - $O(48) \rightarrow O(1)$

# ArrayList Performance (Revisited)

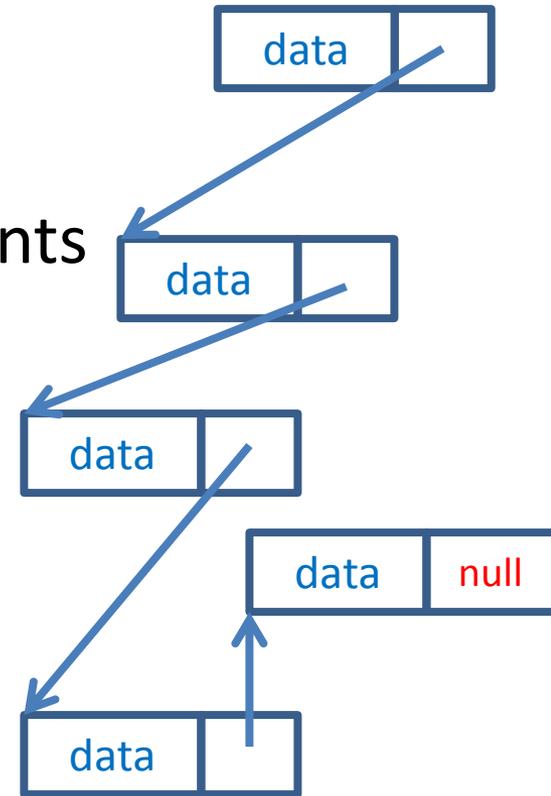
- Fast addition to **end of list**:
  - Fast access to any existing position –  $O(1)$  (like array)
  - Keep extra *capacity* for list growth
    - Fast access includes items in capacity not yet filled –  $O(1)$
  - Capacity management is best left for CSSE230
- Slow inserts to and deletes from middle of list
  - Can get to insert/delete location quickly
  - For insert, shift all items right to accommodate - $O(n)$
  - For delete, shift all items left to fill gap –  $O(n)$

# 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 know where they go
  - Slow access to arbitrary elements



Insertion, per Wikipedia



# 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!