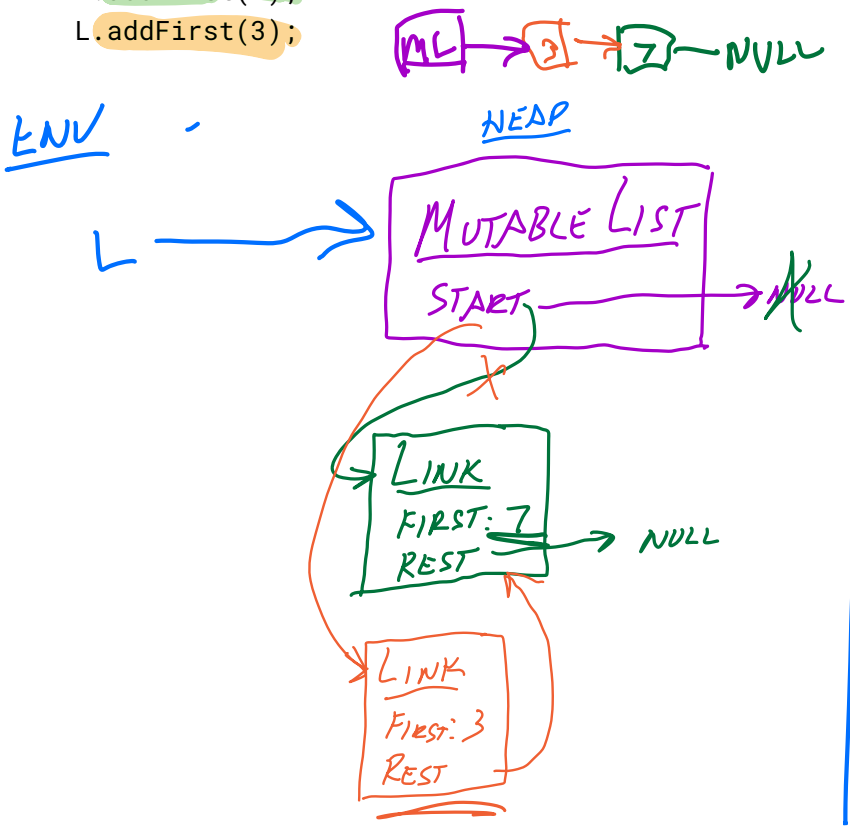
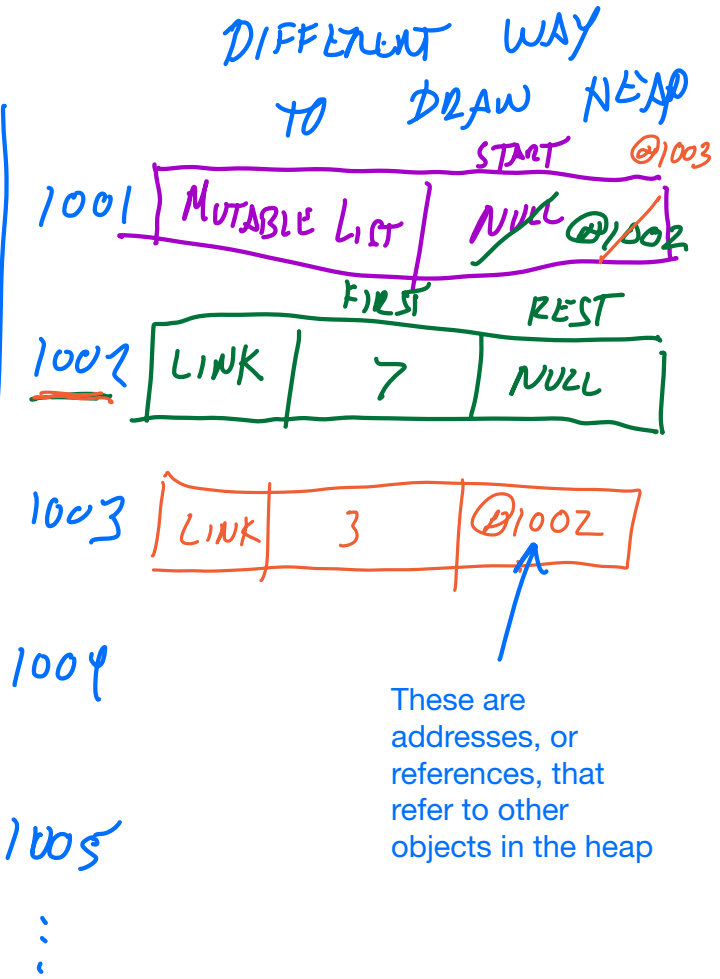
Lecture 10 – Addresses, Equality and ArrayLists

Lesson: Memory Diagrams with Addresses Explicit

```
// the list [3, 7]
MutableList<Integer> L = new MutableList<>();
L.addFirst(7);
L.addFirst(3);
```



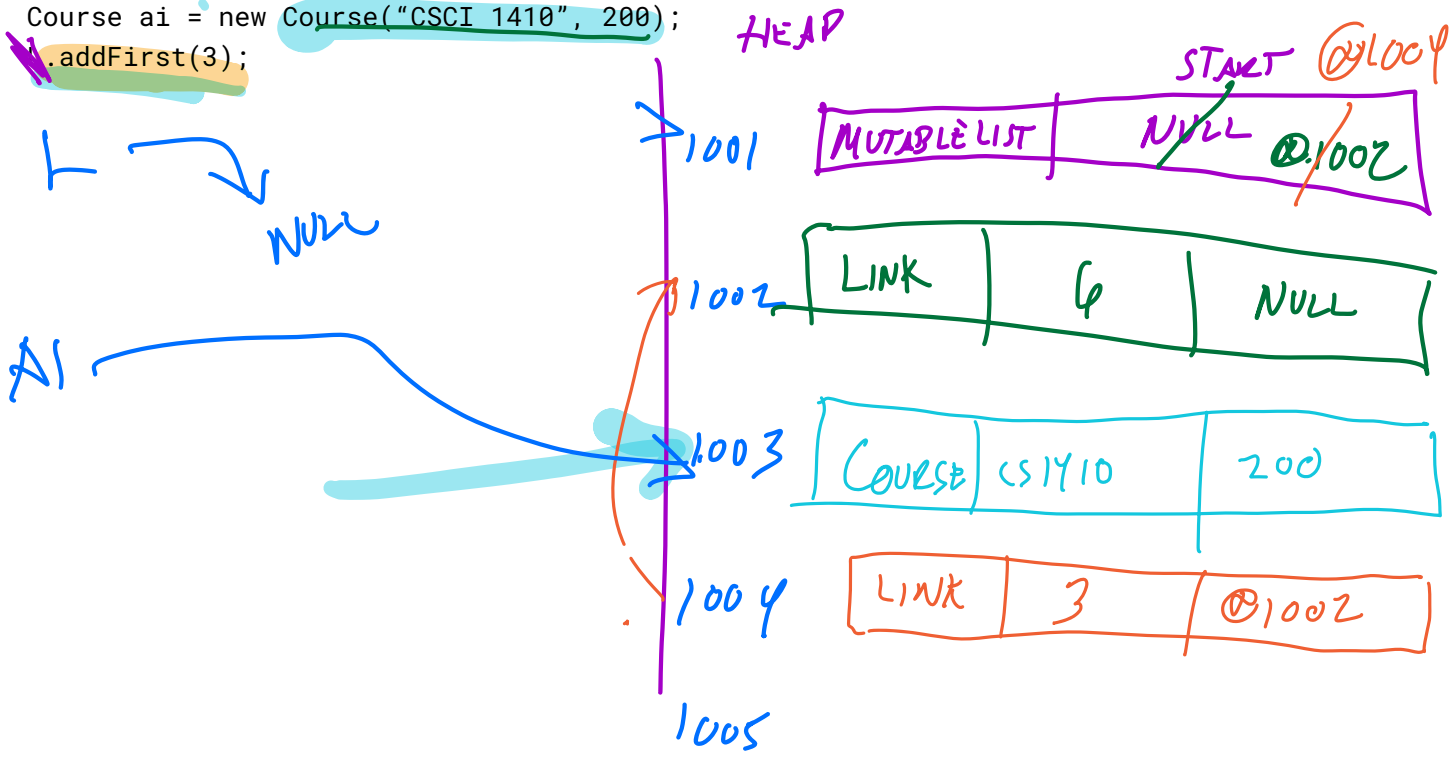
Can think of the heap as a series of addresses
- An address is a label for a specific spot in computer memory
- Every object lives at one address



These are addresses, or references, that refer to other objects in the heap

Activity: Draw the memory diagram with addresses for the following program

```
public void Example2() {
    MutableList<Integer> L = new MutableList<>();
    L.addFirst(6);
    Course ai = new Course("CSCI 1410", 200);
    L.addFirst(3);
}
```



=> When we make new objects ("new") we use the next space in the heap
=> Addresses (or slots) in the heap are used ("allocated") in the order in which the code is run (when we call "new")

Question: What does it mean for lists to be "the same"

```
public static void equalityExample() {
    MutableList<Integer> L1 = new MutableList<Integer>();
    L1.addFirst(6);
    L1.addFirst(8);
    System.out.println("L1 is " + L1);

    MutableList<Integer> L2 = new MutableList<Integer>();
    L2.addFirst(6);
    L2.addFirst(8);
    System.out.println("L2 is " + L2);

    // what do you expect each of these to produce? (what do == and .equals mean?)
    System.out.println(L1 == L2);
    System.out.println(L1.equals(L2));
    System.out.println(L1.toString() == L2.toString());
    System.out.println(L1.toString().equals(L2.toString()));
}
```

DIFF NOTATIONS FOR WHAT IT MEANS FOR OBJECTS TO BE EQUAL?

STRING == STRING

L1 →

L2 →

@1020	MutableList(start: @1022)
@1021	Node(item: 6, next: null)
@1022	Node(item: 8, next: @1021)
@1023	MutableList(start: @1025)
@1024	Node(item: 6, next: null)
@1025	Node(item: 8, next: @1024)

(1) L1 == L2: "are L1 and L2 at the same location in memory". (Also called "Address comparison" "pointer comparison")
=> No, this is false

(2) .equals: Allows programmer to control what equality should mean for this type of object. ("Structural comparison")
=> Programmer would need to write equals method in MutableList (look at all elements, make sure data is the same…).

(3) Comparing strings with == will almost always fail => strings are objects, they live at different locations in memory. (== is okay for int, bool, float, …)
=> Should compare strings with .equals, ie. str1.equals(str2). This checks if the strings have the same characters

Course c1 = new Course("cs200", 80)
Course c2 = new Course("cs200", 84)

Should c1.equals(c2) be true?

=> As programmers, we COULD define .equals to just compare the course name and not the enrollment. This is a decision we would need to make when we write the equals method

Here's an example of writing a Course class with an equals method:

```
public class Course {
   private String name;
   private int enrolled;

   @Override
   // Example of an equals method. Since equals can be called with any
   // other object as the argument, we use type Object for the parameter
   public boolean equals(Object otherObj) {
       if (!(otherObj instanceof Course)) {
           // if otherObj isn't a Course, this and otherObj aren't equal
           return false;
       } else {
           Course otherC = (Course)otherObj; // tell Java otherObj is a Course
           return (this.name.equals(otherC.name));


       }
   }
}
```

In this example, we say that two Course are objects are equal if they have the same name--it's up to the programmer to decide what fields matter!

Since someObj is type Object, we need to tell Java that someObj is really a Course, even though it thinks otherwise.

This syntax is called "casting", and it's used to change how Java thinks about a certain datatype. Beware, though: if otherObj *isn't* the correct type when this code is run, the program will crash!

(In this course, casting is something we'll only need in a few specific situations (like equals() methods), so you don't need to worry about it too much--we'll generally tell you when you need it.)