# Scope, Static Types, and Linked Lists

Discussion 03

# Announcements

Weekly Survey 2- due this Monday 2/1

Lab 2 - due this Tuesday 2/2

Proj 1 Checkpoint - due this Friday 2/5

Lab 3 - due this Friday ⅖

Carefully read the OH guidelines if you attend. OH tickets that do not follow the template *will be removed.*

Read the announcements post on Ed for all the details.

# Content Review

# Pass by Value

**Pass by value** means that when you call a function and give it some arguments, the function called receives an exact copy of those arguments, tied to its own local variables. It is a "shallow" copy, meaning that if a pointer to an object is passed, only the pointer is copied, and the object in memory is not.

```
...
int x = 5;
int[] arr = new int[]{1, 2, 3, 5};
doSomething(x, arr);
...

public void doSomething(int y, int[] a) {
    y = 16;
    a[2] = 4;
}
```

After running the first code block, x = 5 and arr = [1, 2, 4, 5].

# Primitive vs. Reference Types

**Primitive Types** are represented by a certain number of bytes stored at the location of the variable in memory.
*Examples:* byte, short, int, long, float, double, boolean, and char (that's it!)

**Reference Types** are represented by a memory address stored at the location of the variable which points to where the full object is. This memory address is often referred to as a *pointer*.
*Examples*: Strings, Arrays, Linked Lists, Dogs, etc.

**==** compares the information at the location of the variable - so it only works for primitive types unless you are trying to compare the memory address of two object, you want to use **.equals()**

# Static vs. Instance

**Static** variables and functions belong to the whole class.
*Example:* Every 61B Student shares the same professor, and if the professor were to change it would change for everyone.

**Instance** variables and functions belong to each individual instance.
*Example:* Each 61B Student has their own ID number, and changing a student's ID number doesn't change anything for any other student.

# Linked Lists

**Linked Lists** are modular lists that are made up of nodes that each contain a value and a pointer to the next node.
To access values in a Linked List, you must use dot notation.
*Example:* `intList.get(2)`

Instantiation:

```
IntList intList = new IntList();
```

They can be extended or shortened by changing the pointers of its nodes (unlike Arrays).

They can't be indexed directly into like an array, instead the computer has to iterate through all of the nodes up to that point and follow their next pointers.

# Worksheet

# 1 Static Electricity

```
1   public class Pokemon {
2       public String name;
3       public int level;
4       public static String trainer = "Ash";
5       public static int partySize = 0;
6
7       public Pokemon(String name, int level) {
8           this.name = name;
9           this.level = level;
10          this.partySize += 1;
11      }
12
13      public static void main(String[] args) {
14          Pokemon p = new Pokemon("Pikachu", 17);
15          Pokemon j = new Pokemon("Jolteon", 99);
16          System.out.println("Party size: " + Pokemon.partySize);
17          p.printStats()
18          int level = 18;
19          Pokemon.change(p, level);
20          p.printStats()
21          Pokemon.trainer = "Ash";
22          j.trainer = "Brock";
23          p.printStats();
24      }
26      public static void change(Pokemon poke, int
        level) {
27              poke.level = level;
28              level = 50;
29              poke = new Pokemon("Voltorb", 1);
30              poke.trainer = "Team Rocket";
31      }
32
33      public void printStats() {
34          System.out.print(name + " " + level
        + " " + trainer);
35      }
36
37  }
```

# 2 To Do Lists

```
1   StringList L = new StringList(“eat”, null);
2   L = new StringList(“should”, L);
3   L = new StringList(“you”, L);
4   L = new StringList(“sometimes”, L);
5   StringList M = L.rest;
6   StringList R = new StringList(“many”, null);
7   R = new StringList(“potatoes”, R);
8   R.rest.rest = R;
9   M.rest.rest.rest = R.rest;
10  L.rest.rest = L.rest.rest.rest;
11  L = M.rest;
```

# 2 Helping Hand

```
1   public class SLList {
2      Node sentinel;
3
4      public SLList() {
5         this.sentinel = new Node();
6      }
7
8      private static class Node {
9         int item;
10        Node next;
11     }
12
13     public int findFirst(int n) {
14        return ________________________________________;
15     }
16
17     public int findFirstHelper(int n, int index, Node curr) {
18        if (____________________) {
19           return -1;
20        }
21        if (____________________) {
22           return index;
23        } else {
24           return ________________________________________;
25        }
26     }
27  }
```