CS 61B
Spring 2021

Scope, Static, and Linked Lists
Discussion 3: February 01, 2021

1 Static Electricity

```java
public class Pokemon {
    public String name;
    public int level;
    public static String trainer = "Ash";
    public static int partySize = 0;

    public Pokemon(String name, int level) {
        this.name = name;
        this.level = level;
        this.partySize += 1;
    }

    public static void main(String[] args) {
        Pokemon p = new Pokemon("Pikachu", 17);
        Pokemon j = new Pokemon("Jolteon", 99);
        System.out.println("Party size: " + Pokemon.partySize);
        p.printStats()
        int level = 18;
        Pokemon.change(p, level);
        p.printStats()
        Pokemon.trainer = "Ash";
        j.trainer = "Brock";
        p.printStats();
    }

    public static void change(Pokemon poke, int level) {
        poke.level = level;
        level = 50;
        poke = new Pokemon("Voltorb", 1);
        poke.trainer = "Team Rocket";
    }

    public void printStats() {
        System.out.print(name + " " + level + " " + trainer);
    }

}
```

If we alter the static variable by an instance variable of an object, then it should work for all the instance variables of the object!

Static variable

(a) Write what would be printed after the main method is executed.

Party size: 2
Pikachu 17 Ash
Pikachu 18 Team Rocket
Pikachu 18 Brock

(b) On line 28, we set `level` equal to `50`. What `level` do we mean? An instance variable of the `Pokemon` class? The local variable containing the parameter to the `change` method? The local variable in the `main` method? Something else?

The local variable containing the parameter to the change method,

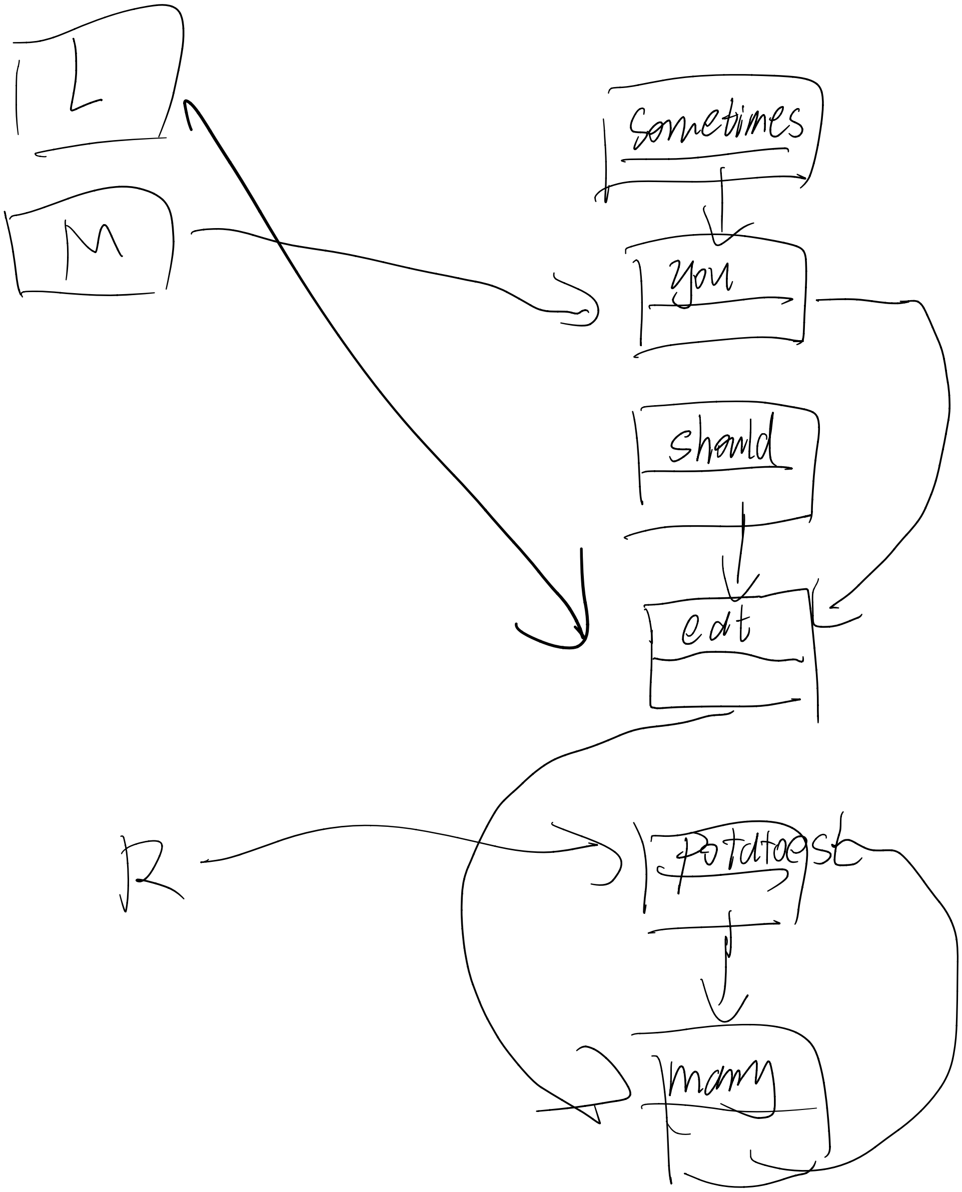
(c) If we were to call `Pokemon.printStats()` at the end of our main method, what would happen?

error

2 To Do List

Draw the box-and-pointer diagram that results from running the following code. A `StringList` is similar to an `IntList`. It has two instance variables, `first` and `rest`.

```
StringList L = new StringList("eat", null);
L = new StringList("should", L);
L = new StringList("you", L);
L = new StringList("sometimes", L);
StringList M = L.rest;
StringList R = new StringList("many", null);
R = new StringList("potatoes", R);
R.rest.rest = R;
M.rest.rest.rest = R.rest;
L.rest.rest = L.rest.rest.rest;
L = M.rest;
```



3 Helping Hand *Extra*

(a) Fill in blanks in the methods `findFirst` and `findFirstHelper` below such that they return the index of the first Node with item `n`, or -1 if there is no such node containing that item.

```
public class SLList {
   Node sentinel;

   public SLList() {
      this.sentinel = new Node();
   }

   private static class Node {
      int item;
      Node next;
   }

   public int findFirst(int n) {
      return findFirstHelper(n, 0, sentinel.next);
   }

   private int findFirstHelper(int n, int index, Node curr) {
      if (curr == null) {
         return -1;
      }
      if (curr.item == n) {
         return index;
      } else {
         return findFirstHelper(n, index+1, curr.next);
      }
   }

}
```

(b) Why do we use a helper method here? Why can't we just have the signature for `findFirst` also have a pointer to the `curr` node, such that the user of the function passes in the sentinel each time?

It's not intuitive for the user to have to pass in 0 and sentinel.next every single time they're calling findFirst(), as it is unrelated to what they're actually requesting. Additionally, it is breaking the abstraction barrier, as it requires our user to understand how this method works under the hood. Finally, if the user didn't understand what to pass in (because again, it's not quite intuitive), they should pass in some random values that will result in an incorrect answer.

Thus, it is bad programming practice to make the user pass in those extra arguments every time. However, we do need a way of keeping track of which node and index we're on as we recurse, so we must make a helper method that can keep track of all that information.