

## 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    }
25
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 }
```

Party size: 2

Pikachu 17 Ash

Pikachu 18 Team Rocket

Pikachu 18 Brock

- (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?

We mean level inside the change method scope

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

This would error, `printStats()` is an instance method but `Pokemon` is the class.

`Pokemon.level` doesn't make sense, and methods are the same

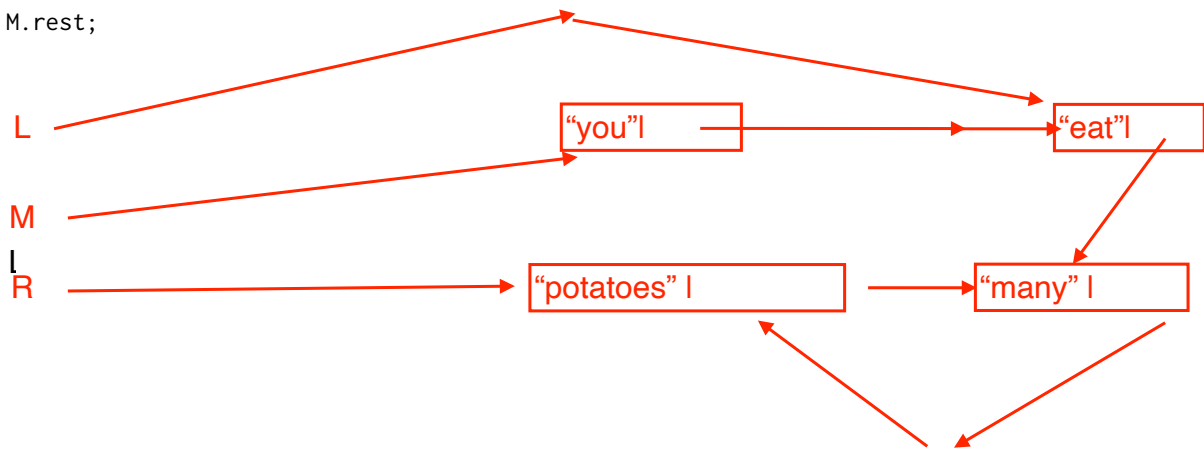
## 2 To Do List

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`.

```

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;

```



### 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.

```

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 findFirstHelper(n, 0, sentinel.next);
15     }
16
17     private int findFirstHelper(int n, int index, Node curr) {
18         if (curr == null) {
19             return -1;
20         }
21         if (curr.item == n) {
22             return index;
23         } else {
24             return findFirstHelper(n, index+1, curr.next);
25         }
26     }
27
28 }
```

- (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?

We have abstracted away the implementation of `SLList`, so the user doesn't know we use a sentinel. Hence, the interface has better UX if we allow users to simply pass in an `n`, so they don't have to worry about implementation details

Also making the helper method public allows the user to make calls to `findFirstHelper` that breaks our implementation eg:  
`findFirstHelper(100, 10000000, sentinel.next.next)`