# 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 = "Cynthia";
        p.printStats();
    }

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

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

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

(b) On line 28, we set `level` equal to `50`. What `level` do we mean?

A. An instance variable of the `Pokemon` object

B. The local variable containing the parameter to the `change` method

C. The local variable in the `main` method

D. Something else (explain)

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

2 Rotate *Extra*

Write a function that, when given an array A and integer k, returns a *new* array whose contents have been shifted k positions to the right, wrapping back around to index 0 if necessary. For example, if A contains the values 0 through 7 inclusive and k = 12, then the array returned after calling rotate(A, k) is shown below on the right:

0	1	2	3	4	5	6	7

$\implies$

4	5	6	7	0	1	2	3
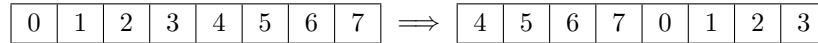
k can be arbitrarily large or small - that is, k can be a positive or negative number. If k is negative, shift k positions to the left. After calling rotate, A should remain unchanged.

Hint: you may find the modulo operator % useful. Note that the modulo of a negative number is still negative (i.e. (-11) % 8 = -3).

```
/** Returns a new array containing the elements of A shifted k positions to the right. */
public static int[] rotate(int[] A, int k) {

    int rightShift = _____;

    if (_____) {

        _____;
    }


    int[] newArr = _____;

    for (_____) {

        int newIndex = _____;

        _____;
    }
    return newArr;
}
```

3   Cardinal Directions

Draw the box-and-pointer diagram that results from running the following code. A DLLStringNode is similar to a Node in a DLList. It has 3 instance variables: prev, s, and next.

```java
public class DLLStringNode {
    DLLStringNode prev;
    String s;
    DLLStringNode next;
    public DLLStringNode(DLLStringNode prev, String s, DLLStringNode next) {
        this.prev = prev;
        this.s = s;
        this.next = next;
    }
    public static void main(String[] args) {
        DLLStringNode L = new DLLStringNode(null, "eat", null);
        L = new DLLStringNode(null, "bananas", L);
        L = new DLLStringNode(null, "never", L);
        L = new DLLStringNode(null, "sometimes", L);
        DLLStringNode M = L.next;
        DLLStringNode R = new DLLStringNode(null, "shredded", null);
        R = new DLLStringNode(null, "wheat", R);
        R.next.next = R;
        M.next.next.next = R.next;
        L.next.next = L.next.next.next;

        /* Optional practice below. */

        L = M.next;
        M.next.next.prev = R;
        L.prev = M;
        L.next.prev = L;
        R.prev = L.next.next;
    }
}
```

4   Gridify

(a) Consider a circular sentinel implementation of an `SLList` of `Node`s. For the first `rows * cols` `Node`s, place the item of each `Node` into a 2D `rows` × `cols` array in row-major order. Elements are sequentially added filling up an entire row before moving onto the next row.

For example, if the `SLList` contains elements $5 \rightarrow 3 \rightarrow 7 \rightarrow 2 \rightarrow 8$ and `rows = 2` and `cols = 3`, calling `gridify` on it should return this grid.

5	3	7
2	8	0

Note: If the `SLList` contains fewer elements than the capacity of the 2D array, the remaining array elements should be 0; if it contains more elements, ignore the extra elements.

Hint: Java's / operator floor-divides by default. Can you use this along with % to move rows?

```
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[][] gridify(int rows, int cols) {
14          int[][] grid = _____;
15          _____;
16          return grid;
17      }
18
19      private void gridifyHelper(int[][] grid, Node curr, int numFilled) {
20          if (_____) {
21              return;
22          }
23
24          int row = _____;
25          int col = _____;
26
27          grid[row][col] = _____;
28          _____;
29
30      }
31  }
```

(b) Why do we use a helper method here at all? i.e., why can't the signature simply be `gridify(int rows, int cols, Node curr, int numFilled)`, omitting `gridifyHelper` entirely?