CS 61B Spring 2024

Scope, Static, Linked Lists, Arrays

Discussion 02: January 29, 2024

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 = "Cynthia";
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("Luxray", 1);
30         poke.trainer = "Team Rocket";
31     }
32
33     public void printStats() {
34         System.out.println(name + " " + level + " " + trainer);
35     }
36 }
```

(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

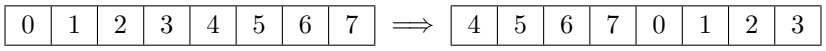
C. The local variable in the `main` method

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:



`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`.

```
1  public class DLLStringNode {
2     DLLStringNode prev;
3     String s;
4     DLLStringNode next;
5     public DLLStringNode(DLLStringNode prev, String s, DLLStringNode next) {
6        this.prev = prev;
7        this.s = s;
8        this.next = next;
9     }
10    public static void main(String[] args) {
11       DLLStringNode L = new DLLStringNode(null, "eat", null);
12       L = new DLLStringNode(null, "bananas", L);
13       L = new DLLStringNode(null, "never", L);
14       L = new DLLStringNode(null, "sometimes", L);
15       DLLStringNode M = L.next;
16       DLLStringNode R = new DLLStringNode(null, "shredded", null);
17       R = new DLLStringNode(null, "wheat", R);
18       R.next.next = R;
19       M.next.next.next = R.next;
20       L.next.next = L.next.next.next;
21
22       /* Optional practice below. */
23
24       L = M.next;
25       M.next.next.prev = R;
26       L.prev = M;
27       L.next.prev = L;
28       R.prev = L.next.next;
29    }
30 }
```

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 $5 \to 3 \to 7 \to 2 \to 8$ 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?