CS 61B Spring 2023 — Scope, Static, Linked Lists, Arrays

Discussion 3: January 30, 2023

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:

$$\begin{array}{|c|c|c|c|c|c|c|c|}\hline 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|c|c|c|c|c|c|}\hline 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 \\ \hline \end{array}$$

`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 module 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;
      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 x 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. Note that the `SLList` may contain more or fewer elements than the capacity of the 2D array:

5	3	7
2	8	0

```java
public class SLList {
    Node sentinel;

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

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

    public int[][] gridify(int rows, int cols) {
      int[][] grid = ___________________________________;
      __________________________________________________;
      return grid;
    }

    private void gridifyHelper(int[][] grid, Node curr, int numFilled) {
      if (_________________________________________________________________________) {
         return;
      }

      int row = ________________________________________;
      int col = ________________________________________;

      ______________________________ = _____________________________;
      __________________________________________________;

    }
}
```

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