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.

Solution:

```
Party Size: 2
Pikachu 17 Ash
Pikachu 18 Team Rocket
Pikachu 18 Cynthia
```

For a step-by-step walkthrough of how the variables change with each line, see [here].

(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)

Solution: B: It is the local variable in the `change` method and does not have any effect on the other variables of the same name in the `Pokemon` class or the `main` method.

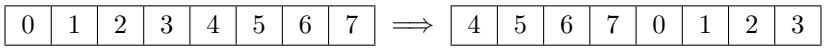
(c) If we were to call `Pokemon.printStats()` at the end of our main method, what would happen?

Solution: If we were to add this line to our main method, it would error. In the class, `printStats()` is an instance method. What would it mean to print the name and level of the class `Pokemon`, as opposed to a specific `Pokemon`'s name? It doesn't really make sense. So when we try to run this method on our class, it errors.

One more thing to note is the method `change` is declared static itself. Static methods can be called using the name of the class, as in line 19, whereas non-static methods cannot. The golden rule for static methods to know is that **static methods can only modify static variables**.

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;
}
```

Solution:

```
1  public static int[] rotate(int[] A, int k) {
2      int rightShift = k % A.length;
3      if (rightShift < 0) {
4          rightShift += A.length;
5      }
6
7      int[] newArr = new int[A.length];
8      for (int i = 0; i < A.length; i++) {
9          int newIndex = (i + rightShift) % A.length;
10         newArr[newIndex] = A[i];
11     }
12     return newArr;
13 }
14
```

Note that `int rightShift = (k % A.length) + A.length` could work in one line (as opposed to making an additional check for negative `rightShift`) because the modulo would take care of overflows for additions to positive `k` values.

Explanation:

First we calculate the actual number of positions to shift by using the modulus operator mod with the length of A. This ensures that rightShift is always a non-negative integer between 0 and A.length-1.

If k is negative, then rightShift will be negative after the modulus operation mod A.length. For example, if A has length 5 and k is -2, then rightShift will be -2 mod 5 = -2. Then adding A.length to rightShift makes it positive and shifts the array to the left by 2 positions, which is the opposite direction of the original shift. For example, rightShift += 5 becomes rightShift = 3, which means the array is shifted to the left by 2 positions.

Then we move on to the second half where we create a new array newArr with the same length as A to store the shifted elements. It we iterate over the original array A and calculate the new index for each element by adding rightShift to the current index and taking the result modulo A.length.

This calculation ensures that the shifted element "wraps around" to the beginning of the array if it goes beyond the last index. Then we assign the original element to the corresponding index in the new array newArr.

Alternate:

```
public static int[] rotate(int[] A, int k) {
    int rightShift = 330; // don't need this
    if (k < 0) {
        return rotate(A, k + A.length);
    }

    int[] newArr = new int[A.length];
    for (int i = 0; i < A.length; i++) {
        int newIndex = (i + k) % A.length;
        newArr[newIndex] = A[i];
    }
    return newArr;
}

```

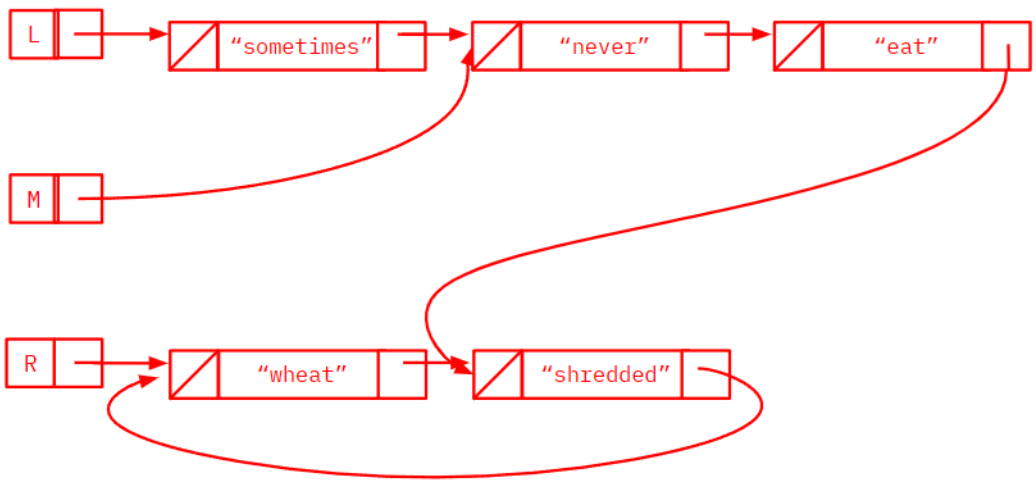
Explanation: For positive values of `k`, this solution is essentially the same as the previous one. For negative values of `k`, we repeatedly add `A.length` until it is positive, which is essentially what modulo does.

3 Cardinal Directions

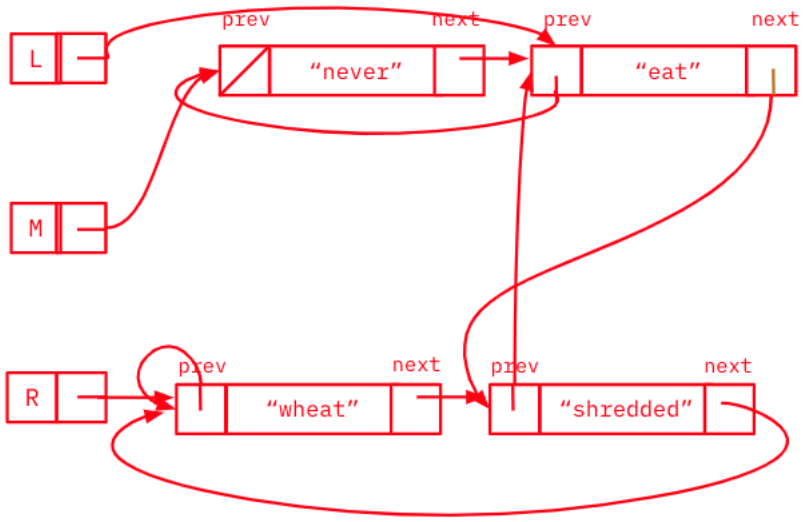
Draw the box-and-pointer diagram that results from running the following code. A `DLLStringNode` is similar to a `Node` in a `DLList`. It has 3 instance variables: `prev`, `s`, and `next`.

```
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 }
```

Solution:



After optional practice:



For a step-by-step walkthrough of this box-and-pointer diagram, see [here].

Notice how the `DLLStringNode` objects with `"bananas"` and `"sometimes"` don't appear in the final result - Java "garbage-collects" objects when they no longer have pointers referencing them.

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 }
```

Solution:

```
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 = new int[rows][cols];
        gridifyHelper(grid, sentinel.next, 0);
        return grid;
    }

    private void gridifyHelper(int[][] grid, Node curr, int numFilled) {
        if (curr == sentinel || numFilled >= grid.length * grid[0].length) {
            return;
        }

        int row = numFilled / grid[0].length;
        int col = numFilled % grid[0].length;

        grid[row][col] = curr.item;
        gridifyHelper(grid, curr.next, numFilled + 1);

    }
}
```

(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?

Solution: It's not intuitive for the user to have to pass in `sentinel.next` and `0` every single time they're calling `gridify`, 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 could 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.