CS 61B Discussion 4: Objects Spring 2020

1 Objects Review

Answer the following questions about the `Avatar` class.

```
1  public class Avatar {
2      public static String electricity; public String fluid;
3  
4      public Avatar(String str1, String str2) {
5          Avatar.electricity = str1;
6          this.fluid = str2;
7      }
8  
9      public static void main(String[] args) {
10         Avatar foo1 = new Avatar("one ", "two");
11         Avatar foo2 = new Avatar("three ", "four");
12         System.out.println(foo1.electricity + foo1.fluid);
13         foo1.electricity = "I declare ";
14         foo1.fluid = "a thumb war";
15         System.out.println(foo2.electricity + foo2.fluid);
16     }
17 }
```

(a) Determine what would be printed after executing the main method of class `Avatar`.

The main method will print the following:

```
three two
I declare four
```

(b) If we changed only line 2 such that `electricity` is an instance variable and `fluid` is a class variable instead, would this code still compile or which other lines would also need to be changed and in what way?

`Avatar` on line 5 will no longer work if `electricity` was no longer static; it would cause a compile-time error because we cannot reference instance variables using a static class reference. But, `this` would still work on line 6 even if `fluid` is made static since an instance variable can be used to reference a static class reference.

(c) Reverting our changes from part (b) and starting from the original code, will adding the following method to class `Avatar` cause any errors during compilation or execution? Why or why not?

```java
public static String getFluid() {
    return fluid;
}
```

The method will cause a compile-time error because we can not reference an instance variable (in this case, `fluid`) from inside a static context.

When the object is not specified (for example, using `fluid` instead of something like `foo1.fluid`) in a field access or method call, Java will use `this.` by default (for example, `this.fluid`). However, since the new method is static, `this` does not exist and therefore an error is thrown.

2 Reversing an Array

Given an array `A`, reverse its elements in place. Do not create any new arrays; this should be a destructive method.

```
/** Destructively reverses A in place. */
public static void reverse(int[] A) {
    for (int i = 0; i < A.length / 2; i++) {
        int temp = A[A.length - i - 1];
        A[A.length - i - 1] = A[i];
        A[i] = temp;
    }
}
```

Extra: Given a square 2D array `B`, reverse the elements along a given `diagonal` in place. `reverseDiagonal` is destructive so you should modify `B` directly. A `diagonal` begins at the leftmost column and continues up and to the right. An N x N array has exactly N diagonals.

For example, let the 2D array on the left be `B`. The value contained within `diagonal = 0` is {1}. The values contained within `diagonal = 1` are {2, 5}. The values contained within `diagonal = 2` are {3, 6, 9}. The values contained within `diagonal = 3` are {4, 7, 10, 13}.

Below, the 2D array on the right has the values along `diagonal = 2` reversed.

1	2	**3**	4
5	**6**	7	8
9	10	11	12
13	14	15	16

$\Longrightarrow$

1	2	**9**	4
5	**6**	7	8
3	10	11	12
13	14	15	16

Assume that `B` contains at least one item (N $\geq$ 0) and that `diagonal` is an integer between `0` and `B.length - 1`, inclusive.

```
/** Destructively reverses the items along the diagonal in B. */
public static void reverseDiagonal(int[][] B, int diagonal) {
    for (int i = 0; i <= diagonal / 2; i++) {
        int temp = B[diagonal - i][i];
        B[diagonal - i][i] = B[i][diagonal - i];
        B[i][diagonal - i] = temp;
     }
}
```

3 Circular Buffer

A circular buffer is a data structure whose contents start at an arbitrary index and continue from there, wrapping around to the beginning upon reaching the end of the buffer. In this case, we will be using an array as our buffer.

Write a function that, when given a full circular array `A` whose contents start at index `i`, returns a *new* array whose contents start at index `0` and maintain the same relative ordering with `k` appended to the end. For example, if `A` has length `8`, contains the values `0` through `7` inclusive, and starts at index `6`, then the array returned after calling `overflow(A, 6, 8)` is shown below on the right (the first item in both buffers are **bolded**):

2	3	4	5	6	7	**0**	1

$\Longrightarrow$

0	1	2	3	4	5	6	7	8

After calling `overflow`, `A` should remain unchanged. Use the `arraycopy` method, which is defined below:

```java
System.arraycopy(Object src, int srcPos, Object dest, int destPos, int length)

/** Returns a new array containing the elements of A starting at index 0
 * with k appended to the end. The contents of the returned array should
 * begin at index 0. The new array should have length = A.length + 1*/
public static int[] overflow(int[] A, int i, int k) {
    int[] B = new int[A.length + 1];
    System.arraycopy(A, i, B, 0, A.length - i);
    System.arraycopy(A, 0, B, A.length - i, i);
    B[A.length] = k;
    return B;
}
```

4 Transposing a 2D Array

The transpose of a 2D array is a new 2D array whose rows are the columns of the original (and vice versa). For example, let A be the 2D array below on the left. The transpose of A is the resulting 2D array below on the right.

$$A = \begin{array}{|c|c|c|c|}\hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ \hline 13 & 14 & 15 & 16 \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|c|c|}\hline 1 & 5 & 9 & 13 \\ \hline 2 & 6 & 10 & 14 \\ \hline 3 & 7 & 11 & 15 \\ \hline 4 & 8 & 12 & 16 \\ \hline \end{array} = transpose(A)$$

Given a square 2D array A, destructively transpose A.

```java
/** Destructively transposes A. Assume that A is square. */
public static void transpose(int[][] A) {
    for (int i = 0; i < A.length; i++) {
        for (int j = i; j < A[i].length; j++) {
            int temp = A[j][i];
            A[j][i] = A[i][j];
            A[i][j] = temp;
        }
    }
}
```