Exam 2 SOLUTIONS

Total: 1000 points

CSCI 110 Section 1

Wednesday, October 26, 2016

1) Please fill in the following table. Write yes/no for first two rows and call methods using the instance names provided for the last three rows. Put n/a if something doesn't exist. [50 points]

	ArrayList	**array**	**String**
Instance name	list	arr	s
Can increase length?	yes	no	no
Mutable?	yes	yes	no
Length	list.size()	arr.length	s.length()
Get element at index i	list.get(i)	arr[i]	s.charAt(i) OR s.substring(i, i+1)
Set element at index i to the value 45	list.set(i, 45)	arr[i] = 45	n/a

2) What's printed when mystery(5) is called? [50 points]

```
void mystery(int n) {
    System.out.println("tiger");
    if (n <= 1) {
        System.out.println("newt");
        return;
    }
    mystery(n / 2);
    System.out.println(n);
}
```

tiger
tiger
tiger
newt
2
5

Here's the sequence of function calls:

- Inside mystery(5), n == 5
- Print "tiger"
- Check base case; condition is false
- Call mystery(5/2)
    - Inside mystery(2), n == 2
    - Print "tiger"
    - Check base case; condition is false
    - Call mystery(2/2)
        - Inside mystery(1), n == 1
        - Print "tiger"
        - Check base case, condition is true
        - Print "newt"
        - Return
    - Print 2
- Print 5

For questions 3 and 4:

```
class Tiger {
    private double hunger;

    public Tiger(double hunger) {
        this.hunger = hunger;
    }

    public double getHunger() {
        return hunger;
    }

    public boolean willEatAntelope() {
        return hunger > 50.0;
    }

    public void eatFood(double food) {
        hunger -= food;
    }
}
```

3) What's printed after the following code is run? [50 points]

```
Tiger t1 = new Tiger(100.0);
System.out.println(t1.willEatAntelope());
t1.eatFood(50.0);
System.out.println(t1.willEatAntelope());
System.out.println(t1.getHunger());
t1.eatFood(50.0);
System.out.println(t1.getHunger());
```

true
false
50.0
0.0

Tiger t1 starts out with hunger 100.0 so it's willing to eat an antelope at first. However, then it eats something and its hunger becomes 50.0. When we ask whether it will eat an antelope a second time, the answer is no. Then we print out the hunger (50.0), eat 50.0 more food, and then print out the hunger (now 0.0).

Say the following two functions are defined along with the previous class.

```
void nature(Tiger t) {
    t.eatFood(20.0);
}

Tiger nurture(Tiger t) {
    t = new Tiger(90.0);
    return t;
}
```

4) What's printed after the following code is run? [50 points]

```
Tiger t2 = new Tiger(60.0);
System.out.println(t2.willEatAntelope());
nature(t2);
System.out.println(t2.willEatAntelope());
t2 = nurture(t2);
System.out.println(t2.willEatAntelope());
```

true
false
true

Tiger t2 starts out with hunger 60.0 so it's willing to eat an antelope at first.

We call the function nature() with t2 as a parameter, and nature modifies t2 by having it eat 20.0 worth of food. The second time we print whether it will eat an antelope, the result is now false.

Finally, we call nurture() on t2 and assign the value back into t2. Inside nurture(), a new Tiger is created with hunger 90.0 and this new Tiger is returned. When the nurture() function returns, that new Tiger is assigned into t2. t2 now references an entirely new Tiger with hunger 90.0. When we print whether it will eat an antelope, the result is true.

5) Please number the blanks from 1 through 9 in the order that each expression is evaluated. A blank may have two numbers. [100 points]

```
class SomeClass {
    static void someFn() {

    __8__   System.out.println("ok");

    }

    static void otherFn() {

    ____    System.out.println("now");

    }

    public static void main(String[] args) {

    __1__   System.out.println("goodbye");

                 __2__      _3__6_  __5__
            for (int i = 0; i < 1; i++) {
    __4__       System.out.println(25);
            }

    __7__   someFn();

    __9__   System.out.println("done");

    }
}
```

Java programs start with the main function. With a for loop, the first thing that happens is the initializer part, then the conditional gets checked. After that, the body runs. Then the variable gets incremented, and the conditional gets checked again.

otherFn never gets called.

6) Please complete the constructor, advance(), rewind(), and getCurrentTape() methods for the VCR class. [100 points]

```
class VCR {
    private double tapePosition;
    private String tapeName;

    /**
     * Constructs a new VCR with a given tapeName. Initializes
     * tapePosition to 0.
     */
    public VCR(String tapeName) {
        this.tapeName = tapeName;
        this.tapePosition = 0;
    }

    /**
     * Advances tapePosition by 10.
     */
    void advance() {
         tapePosition += 10;
    }

    /**
     * Rewinds tapePosition to 0.
     */
    void rewind() {
        tapePosition = 0;
    }

    /**
     * Returns the name of the current tape.
     */
    String getTapeName() {
        return tapeName;
    }
}
```

Both advance() and rewind() modify the object, but don't need to return anything.

7) Write a function to translate a pixel intensity to a description. Pixel intensities can be between 0 and 255. Intensities between 0 and 85 should be described as "light", between 86 and 170 should be described as "medium", between 171 and 255 should be described as "dark". Intensities outside of this range should result in "INVALID_INTENSITY". [100 points]

Return type? String, the description
Parameters? int, the pixel intensity as a number

```
String getDescription(int intensity) {
    if (intensity >= 0 && intensity <= 85) {
        return "light";
    } else if (intensity >= 86 && intensity <= 170) {
        return "medium";
    } else if (intensity >= 171 && intensity <= 255) {
        return "dark";
    }
    return "INVALID_INTENSITY";
}
```

8) Say there's a pre-defined function with the call signature "boolean isHappy(int num)" that returns true if a number is happy. Write a function that returns the sum of the happy numbers between (including) 7 and (excluding) 700. [100 points]

Return type? int, the sum of the happy numbers
Parameters? none since we always go from 7 to 700

```
int getHappySum() {
    int sum = 0;
    for (int i = 7; i < 700; i++) {
        if (isHappy(i)) {
            sum += i;
        }
    }
    return sum;
}
```

9) Write a function named oppositeDay that inverts all of the values in an ArrayList<Boolean> i.e. turns true to false and false to true. This function shouldn't return anything. [100 points]

Return type? void; we're modifying an existing ArrayList
Parameters? the ArrayList<Boolean> we want to modify

```
void invert(ArrayList<Boolean> bools) {
    for (int i = 0; i < bools.size(); i++) {
        boolean value = bools.get(i);
        bools.set(i, !value);
    }
}
```

10) Write a function that takes an array and returns a new array with two copies of the original, one after the other. For instance, if this function was given [6, 2, 3, 4], it should return [6, 2, 3, 4, 6, 2, 3, 4]. [100 points]

I've chosen to do this with arrays of boolean since the type wasn't specified.

Return type? the repeated boolean[]
Parameters? the boolean[] we want to repeat

```
boolean[] seeingDouble(boolean[] bools) {
    boolean[] newBools = new boolean[bools.length * 2];
    for (int i = 0; i < bools.length; i++) {
        // Puts bools[i] in the first spot in the repeated version
        newBools[i] = bools[i];

        // Puts bools[i] in the second spot in the repeated version
        newBools[i + bools.length] = bools[i];
    }
    return newBools;
}
```

11) Say that monthly listener counts on Spotify are represented as an array of long. We want to take this array and find how many people listen to the most and least popular artists. Write a function named findDrakeAndMeek that finds both the min and max of an array of long and returns the sum of the min and max. [100 points]

Return type? long, the sum of the min and max
Parameters? long[], the array of monthly listener counts

This is a little tricky since we're getting both the min and the max. You could do this with two separate for loops, or you can find both using just one.

The function starts out assuming both the min and the max are the first element, which is true for an array with one element. Then it tries to find larger and smaller elements to put into max and min.

```java
long findDrakeAndMeek(long[] counts) {
    long min = counts[0];
    long max = counts[0];
    for (int i = 1; i < counts.length; i++) {
        if (counts[i] > max) {
            max = counts[i];
        } else if (counts[i] < min) {
            min = counts[i];
        }
    }
    return min + max;
}
```

12) Write a function named bullseye that takes a 2D array of ints. The outer array always has an odd number of elements. Each inner array always has the same odd number of elements. bullseye should set the middle element of the middle array to be 1. [100 points]

Example:

```
int[][] grid = {{0, 0, 0, 0, 0},
                 {0, 0, 0, 0, 0},
                 {0, 0, 0, 0, 0},
                 {0, 0, 0, 0, 0},
                 {0, 0, 0, 0, 0}};

bullseye(grid);

// grid now contains:
//             {{0, 0, 0, 0, 0},
//              {0, 0, 0, 0, 0},
//              {0, 0, 1, 0, 0},
//              {0, 0, 0, 0, 0},
//              {0, 0, 0, 0, 0}}
```

Return type? void; we're modifying an existing 2d array
Parameters? int[][], the 2d array we're modifying

This solution is deceptively simple. Let's work through this piece by piece:

- Inside the first set of square brackets, ints.length gives us the number of inner arrays. This means ints.length/2 gives us the index of the middle inner array of int. Thanks to integer division, 1/2 == 0 and 3/2 == 1 and 5/2 == 2 and so on. Dividing by two always gives us the index of the middle inner array.
- Inside the second set of square brackets, ints[0].length gives us the length of the first inner array. The question tells us that all of the inner arrays have the same length, so we take the length of the first one. And with the same principle as before, ints[0].length/2 gives us the middle index of an inner array (which is the same for all inner arrays).
- Finally, we assign 1 to the middle index of the middle inner array.

```
void bullseye(int[][] ints) {
    ints[ints.length/2][ints[0].length/2] = 1;
}
```

13) Write a function named makeWeirdCase that takes a String and returns a String with the first letter lowercase, the second letter uppercase, the third letter lowercase, and so on. For example, if the function is given "class", it should return "cLaSs". [extra credit, 100 points]

Return type? String, the result of making our input have the weird case
Parameters? one String, the String we want to create a weird version of

We're building up the weird version letter by letter, starting with a lowercase letter. isLower is a boolean that determines the case of the next letter we put in, and it gets inverted at the end of each iteration.

```java
String makeWeirdCase(String s) {
    StringBuilder builder = new StringBuilder();
    boolean isLower = true;
    for (int i = 0; i < s.length(); i++) {
        if (isLower) {
            builder.append(s.substring(i, i+1).toLowerCase());
        } else {
            builder.append(s.substring(i, i+1).toUpperCase());
        }
        isLower = !isLower;
    }
    return builder.toString();
}
```

14) Write a function that determines if a given string can be obtained by one concatenation of some string to itself. For example, "lionlion" can be obtained by appending "lion" to itself. "www" is an example of a String for which there's no way to do this. [extra credit, 100 points]

Return type? boolean; we're determining *whether* something is true
Parameters? the String we're analyzing

This solution is deceptively simple.

- Take the example "haha" which can be obtained by one concatenation of "ha" to "ha". The length of "haha" is 4, and s.length() / 2 == 2.
    - s.substring(0, 2) gives us "ha".
    - s.substring(2, 4) gives us "ha".
    - "ha".equals("ha") is true, so the function returns true
- You might have noticed that the length of the string must be even, so you might have written an if statement to return false if the length is odd. However, our single comparison is enough to catch this case. Consider "www" as an example. s.length() is 3 and s.length() / 2 == 1.
    - s.substring(0, 1) gives us "w".
    - s.substring(1, 3) gives us "ww".
    - "w".equals("ww") is false, which is what we expected
- What about edge cases, such as the empty string? For an empty string, s.length() / 2 is zero and s.length() is zero. An empty string is equivalent to an empty string so the function returns true. And we expect it to be true since appending the empty string to itself creates the empty string.

Using conditionals would have been fine, but this is a straightforward answer that works just as well.

```java
boolean isTandem(String s) {
    String firstHalf = s.substring(0, s.length() / 2);
    String secondHalf = s.substring(s.length() / 2, s.length());
    return firstHalf.equals(secondHalf);
}
```