

Name: _____

USC NetID (e.g., ttrojan): _____

CS 455 Midterm Exam 1

Fall 2017 [Bono]

Thursday, Sep. 28, 2017

There are 6 problems on the exam, with 55 points total available. There are 10 pages to the exam (5 pages **double-sided**), including this one; make sure you have all of them. If you need additional space to write any answers or scratch work, pages 8, 9 and 10 are left blank for that purpose. If you use those pages for answers you just need to direct us to look there. *Do not detach any pages from this exam.*

Note: if you give multiple answers for a problem, we will only grade the first one. Avoid this issue by labeling and circling your final answers and crossing out any other answers you changed your mind about (though it's fine if you show your work).

Put your name and USC username (a.k.a., NetID) at the top of the exam. Also put your NetID at the top right of the front side of each page of the exam. Please read over the whole test before beginning. Good luck!

Selected methods of Java `Rectangle` class:

```
Rectangle(int x, int y, int width, int height)
```

Constructs rectangle object whose upper-left corner is (x, y) and whose width and height are specified by the arguments of the same name.

```
Rectangle(Rectangle anotherRect)
```

Constructs rectangle object with the same x, y, width, and height as `anotherRect` (copy constructor).

```
void translate(int dx, int dy)
```

Changes x and y values of this rectangle by dx and dy, respectively. I.e., if this rectangle had upper-left coordinates (x, y), its value after the call is a rectangle with upper-left coordinates (x+dx, y+dy) [this is a mutator]

Reminder of some Unix shell commands and other shell syntax:

```
ls cd cp mv rm mkdir rmdir more diff
```

```
~ * .. < > & .
```

Problem 1 [11 pts. total]

Consider the following program. Note: it uses the Java `Rectangle` class – more information about `Rectangle` methods on the cover of the exam.

```
public class Prob1 {  
    public static void main(String[] args) {  
        Rectangle rect = new Rectangle(6, 8, 20, 30);  
        Rectangle rect2 = new Rectangle(rect);  
        Rectangle rect3 = rect;  
  
        rect.translate(3, 7);  
        rect3.translate(5, 10);  
        System.out.print(rect.getX() + " " + rect.getY() + ",");  
        System.out.print(rect2.getX() + " " + rect2.getY() + ",");  
        System.out.println(rect3.getX() + " " + rect3.getY());  
    }  
}
```

Ⓐ

Part A [3]. Circle all of the following boolean expressions that are true at the point in the code labeled Ⓐ above.

1. `rect.equals(rect2)`

2. `rect2.equals(rect3)`

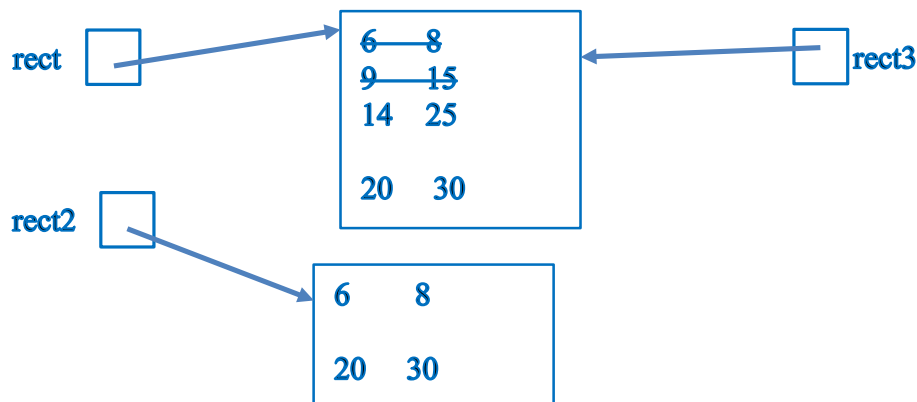
3. `rect2.equals(rect3)`

4. `rect == rect2`

5. `rect == rect3`

6. `rect2 == rect3`

Part B [5]. In the space below, draw a box-and-pointer diagram (a.k.a., memory diagram) showing all object variables, objects, and their state as they have changed during *the whole code sequence* (i.e., including statements *after* point Ⓐ).



Part C [3]. What is printed by the code? `14 25, 6 8, 14 25`

Problem 2 [2 pts]

Consider the following version of the `Student` class (not all methods shown):

```
public class Student {
    private String name;
    private int totalScore;
    public Student(String name) {
        name = name;
        totalScore = 0;
    }
    . . .
}
```

What value does the instance variable `name` have in the `Student` object `pupil` after the following call to the constructor: (Hint: read the class code closely.)

```
Student pupil = new Student("Donald");
```

`null`

Problem 3 [3 pts]

Suppose your home directory on `aludra` only contains subdirectories `pa1`, `pa2`, `lab1`, `lab2`, `lab3`, etc. where you stored all the files for the assignments. You now decide you want to change your file organization so that all of your coursework for CSCI 455 will be in a new subdirectory in your home directory called `cs455` and *that* directory will contain those individual subdirectories for each lab or programming assignment you have done so far. **Give a sequence of 3 or fewer Unix commands to change your file organization as described (assume the current working directory is your home directory).** Note: The front page of this exam has a reminder of the names of some common Unix commands and other syntax.

`mkdir cs455`

`mv lab* cs455`

`mv pa* cs455`

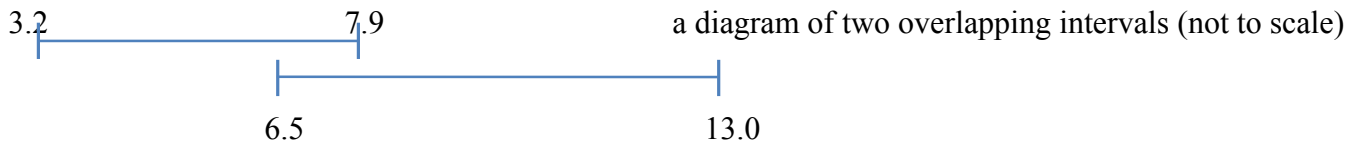
alternate solution:

`mkdir cs455`

`mv * cs455`

Problem 4 [18 pts]

Implement the class `Interval`, which represents a one-dimensional closed interval on a number line. Closed means that the endpoints of the interval are also included in the interval, e.g., `[3.2, 7.9]`



Notes: (1) This class does no I/O. (2) For this problem you do not have to worry about round-off errors. (3) A Java rule: When a class's method has an explicit parameter of the same type as the class itself, it can access the private data of that explicit parameter (e.g., `other` parameters of type `Interval` in `overlaps` and `intersection` methods below).

See the class and method comments and headers for the exact interface. Space for your answer starts here and continues onto the next page.

```
// A class to represent a one-dimensional closed interval
public class Interval {
    // space for instance variables here:
```

```
    private double low;
    private double high;
```

```
    // Creates the interval from low to high
    // PRECONDITION: low <= high
    public Interval(double low, double high) {
```

```
        this.low = low;
        this.high = high;
```

```
    }
```

```
    // Returns the low bound of the interval.
    public double low()
```

```
        return low;
```

```
    }
```

```
    // Returns the high bound of the interval.
    public double high()
```

```
        return high;
```

```
    }
```

[class definition continued next page]

Problem 4 (cont.)

Complete the implementation of `Interval` – details of this problem are on the previous page.

```
// Returns true iff this Interval contains the specified point.
public boolean contains(double point) {
```

```
    return low <= point && point <= high;
```

```
}
```

```
// Returns true iff this Intervals and the specified Interval overlap (i.e.,
// contain one or more points in common).
public boolean overlaps(Interval other) {
```

```
    if ((this.low > other.high) || (this.high < other.low)) {
        return false;
```

```
    }
```

```
    return true;
```

alternate solution:

```
    return (this.low <= other.high) && (this.high >= other.low);
```

```
}
```

```
// Returns the Interval that is the intersection of this Interval with the
// specified Interval. If they do not intersect, returns null.
public Interval intersection(Interval other) {
```

// alternate check for no intersects: call overlaps and return on false before the other code

```
    double maxLow = Math.max(this.low, other.low);
```

// or could use: if (this.low >= other.low) { maxLow = this.low; }, etc.

// (w/o call to Math.max)

```
    double minHigh = Math.min(this.high, other.high);    // or if-else
```

```
    if (maxLow > minHigh) { return null; } // they don't intersect
```

```
    return new Interval(maxLow, minHigh);
```

```
}
```

```
}
```

Problem 5 [6 pts. total]

The following method doesn't work as desired. The method comment describes what it is *supposed* to do.

```
/**
 * Computes the sum of a sequence of non-negative scores read in from the given
 * Scanner, using -1 as a sentinel. The method assumes the score data is in the
 * correct format.
 *
 * Ex1:  input: 3 8 8 -1      returns 19
 * Ex2:  input: 5 -1         returns 5
 * Ex3:  input: -1           returns 0
 *
 * @param    in the Scanner to read from
 * @returns  the sum of the scores read
 */
public static int sumScores(Scanner in) {

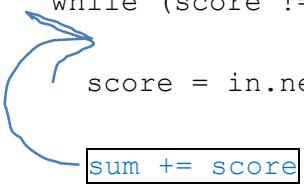
    int sum = 0;

    int score = in.nextInt();

    while (score != -1) {
        score = in.nextInt();
        sum += score;
    }

    return sum;

}
```



Part A [2]. Show sample input and return value for this buggy method such that the method doesn't return the correct value.

example input: 3 8 8 -1

corresponding return value: 15

Part B [4]. Fix the code above. Do not rewrite the whole method, but rather make your changes right into the code above, using arrows to show where any new code should be inserted, ~~crossing out~~ code that you would get rid of, etc.

Problem 6 [15 points]

Write the static Java `int` method, `numTriple3s`, which takes an array of numbers and returns the number of triple 3s it contains, that is, the number of times three adjacent 3s appear in the array. Any particular 3 in the array can only be counted as part of one such run (see 4th and 5th examples below).

Examples

<u>vals</u>	<u>numTriple3s(vals)</u>
<code>[3, 3]</code>	<code>0</code>
<code>[5, 5]</code>	<code>0</code>
<code>[0, 3, 3, 5, 3, 3, 3, 7]</code>	<code>1</code>
<code>[0, 3, 3, 3, 3, 3, 3, 8]</code>	<code>2</code>
<code>[0, 3, 3, 3, 3, 3, 40, 8]</code>	<code>1</code>
<code>[3, 3, 3]</code>	<code>1</code>
<code>[3, 3, 3, 1, 2, 3, 3, 3]</code>	<code>2</code>
<code>[3, 4, 3, 7, 5, 3]</code>	<code>0</code>
<code>[]</code>	<code>0</code>

```
public static int numTriple3s(int[] vals) {
    int count = 0;
    for (int i = 0; i < vals.length - 2; i++) {
        if (vals[i] == 3 && vals[i+1] == 3 && vals[i+2] == 3) {
            count++;
            i += 2;
            // note: loop will increment this by one more before next iter
        }
    }
    return count;
}
```

continued on the next page . . .

Problem 6 alternate solution:

```
int count = 0;    // count of the number of triple-3s
int currRun = 0;  // number of 3s in current run
for (int i = 0; i < vals.length; i++) {
    if (vals[i] == 3) {
        currRun++;
        if (currRun == 3) { // found 3 in a row
            currRun = 0;    // reset the counter for current run
            count++;        // count the triple
        }
    }
    else { // it wasn't a 3 (and we know count < 3 here)
        currRun = 0;    // not in a run anymore.
    }
}
return count;
```

there were variations on these solutions, plus a few other different approaches that also work.

Extra space for answers or scratch work.

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.

Extra space for answers or scratch work (cont.)

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.

Extra space for answers or scratch work (cont.)

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.