

Name: _____

CSSE 220—Object-Oriented Software Development

Exam 1, October 1, 2013

This exam consists of two parts. Part 1 is to be solved on these pages. If you need more space, please ask your instructor for blank paper. After you finish Part 1, please turn in your Part 1 answers and then open your computers.

Part 2 is to be solved using your computer. You will need network access to download template code and upload your solution for part 2. Please disable IM, email, and other such communication programs before beginning the exam. Any communication with anyone other than the instructor or a TA during the exam, *may result in a failing grade for the course*.

Allowed Resources on Part 1: You are allowed one 8.5 by 11 sheet of paper with notes of your choice. This section is *not* open book, open notes, and you cannot use your computer for this part.

Allowed Resources on Part 2: Open book, open (printed physical) notes, and computer. Limited network access. You may use the network only to access your own files, the course Moodle site and web pages, the textbook's site, Sun's Java website, and Logan Library's Safari Tech Books Online.

Parts 1 and 2 are included in this document. You should read over all questions before beginning work, but *you must turn in part 1 before accessing resources for part 2*.

	Problem	Poss. Pts.	Earned
	1	9	_____
	2	6	_____
	3	8	_____
	4	5	_____
	5	6	_____
	6	6	_____
	Paper Part Subtotal	40	_____
	Computer Part A	Points	Earned
	findLargest	6	_____
	multiplyImaginaryByReal	6	_____
	combineStrings	6	_____
	findFirstLetterMatches	6	_____
	count4s	6	_____
	Computer Part B		
	Example 1 functionality	10	_____
	Example 2 functionality	10	_____
	Example 3 functionality	10	_____
	Computer Part Subtotal	60	_____
	Total	100	_____

Part 1—Paper Part

The next several questions all refer to a `ImaginaryNum` class. Below is a listing of this class showing its fields, constructors, accessor methods, and mutator methods. The javadocs are omitted to save space.

```
public class ImaginaryNum {
    private double real, imaginary;

    public ImaginaryNum(double real, double imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    public void setReal(double real)
    {
        this.real = real;
    }

    public double getReal()
    {
        return this.real;
    }

    public ImaginaryNum addAndCreate(double real)
    {
        ImaginaryNum add = new ImaginaryNum(real + this.real, this.imaginary);
        return add;
    }

    public void setImaginary(double imaginary)
    {
        this.imaginary = imaginary;
    }

    public String toString()
    {
        return "r=" + this.real + " i=" + this.imaginary;
    }
}
```

1. (9 points) Below are several code snippets that use the `ImaginaryNum` class. For each snippet, first *draw a box-and-pointer diagram* showing the result of executing it. Then *give the output* of the print statement at the end of the snippet.

```
int val = 7;
ImaginaryNum num = new ImaginaryNum(val, val);
val = 8;
System.out.println(num.toString());
```

(a) Output: _____
Diagram: _____

```
ImaginaryNum one = new ImaginaryNum(1,0);
ImaginaryNum two = one;
two.setReal(2);
ImaginaryNum unknown = two.addAndCreate(1);
System.out.println(unknown.getReal());
```

(b) Output: _____
Diagram: _____

```
ImaginaryNum[] n = new ImaginaryNum[4];
ImaginaryNum otherNum = new ImaginaryNum(1,1);
for(int i = 0; i < 3; i++) {
    n[i] = otherNum;
}
System.out.println(n[3]);
```

(c) Output: _____
Diagram: _____

2. (6 points) For each code snippet below predict its output. (You do *not* need to draw a diagram, but you may if it might help you.)

```
String[] data = new String[10];  
data[0] = "Robot";  
data[1] = "Pirate";  
data[2] = "Ninja";  
System.out.println(data.length);
```

(a) Output: _____

```
double percent = 0.70;  
int intPercent = (int) percent;  
double result = (double) (100 * intPercent);  
System.out.println(result);
```

(b) Output: _____

```
ArrayList<String> al = new ArrayList<String>();  
al.add("Hello");  
al.add("World");  
System.out.println(al.get(1));
```

(c) Output: _____

3. (8 points) For each for loop below, write down how many times its body will execute, or indicate that we can't tell from the information given.

```
for (int i = 0; i <= 6; i++) {  
    // loop body elided  
}
```

(a) Answer: _____

```
int maxTimer = 10;  
while (maxTimer > 3) {  
    maxTimer--;  
}
```

(b) Answer: _____

```
int[] data = new int[10];  
for(int data2 : data) {  
    // loop body elided  
}
```

(c) Answer: _____

```
while(true) {  
    System.out.println("in loop body");  
    break;  
}
```

(c) Answer: _____

4. (5 points) Explain the difference in behavior between the following two code examples (if any). *If they are functionally different, explain why they are different AND which one you would prefer. If they are functionally the same, just write "They are the same."*

```
System.out.println("Password?");
Scanner in = new Scanner(System.in);
String secretWord = "secret";
if(secretWord.equals(in.next())) {
    System.out.println("You won!");
}
```

and

```
System.out.println("Password?");
Scanner in = new Scanner(System.in);
String secretWord = "secret";
if(secretWord == in.next()) {
    System.out.println("You won!");
}
```

5. (6 points) For each of the code snippets below, there is a bug that causes the code to work incorrectly (i.e. the problem is just not weird style). First *circle* the bug, then *write code that fixes the problem* (if only one line is wrong, you only have to rewrite that one line).

```
ImaginaryNum seven;
seven.setReal(49.0/7);
```

Circle the bug in the code above and write code that fixes the problem:

```
String miss = "Mississippi";
miss.replace("i", "*");
System.out.printf("Mississippi without 'i's: %s\n", miss);
```

Circle the bug in the code above and write code that fixes the problem:

6. (6 points) *Unit Testing*: Consider the documentation for the method below.

```
/**
 * Repeats the given string a given number of times
 *
 * For example, repeatString("hello",3) returns "hellohellohello"
 *
 * @param input
 * @param timesToRepeat
 * @return input string repeated timesToRepeat times
 */
public static String repeatString(String input, int timesToRepeat) {
    // body code elided
}
```

In the table below, give three sets of values that would constitute a good test set for this method. For each argument you list, say briefly why it should be in the unit test.

Argument to method	Expected Result	Why is this a good test?
a)		
b)		
c)		

Part 2—Computer Part

Instructions. You must actually get these problems working on your computer. Almost all of the credit for the problems will be for code that actually works. There are several different small methods to write, so you can get a lot of partial credit by getting some of them to work. If you get every part working, comments are not required. If you do not get a method to work, comments may help me to understand enough so I can give you (a small amount of) partial credit.

Begin part 2 by checking out the project named *Exam1* from your course SVN repository. (Ask for help immediately if you are unable to do this.)

When you have finished a problem, and more frequently if you wish, **submit your code by committing it to your SVN repository.** We will check commit logs, so you must be careful not to commit anything after the end of the exam. For grading, we will ensure that the included JUnit tests have not been changed.

Problem Descriptions

Part A: 5 Small Problems. (30 points) Implement the code for the 5 functions in PartA.java. Instructions are included in the comments of each function. Unit tests are included in PartATests.java.

Part B: Checkerboard (30 points)

Read over all these instructions carefully. Make sure you understand completely what functionality you have to implement before you start coding. Ask if any part of the instructions are unclear.

Implement the Checkerboard class. The checkerboard class represents a board made up of rows and columns of rectangles. Checkboards can be positioned at various x and y coordinates, can have various widths and heights, and can have 2 to any number of colors in the board pattern.

The way colors in the checkerboard should work is that they form a repeating pattern as you read across the row: white, black, white, etc. When you get to a new row you just continue with the pattern. The pattern always starts with white then black. If new colors are added they are inserted in the pattern at the end. So if you add green the pattern becomes white, black, green, white, black, green, etc. If you add green and then red the pattern becomes white, black, green, red, white, etc.

You are allowed to implement the Checkerboard's functions in any order you wish. However, to make your life easier we have set up 3 separate examples that make for a good order to implement functionality in. Bear in mind that as you implement later examples earlier examples should keep working.

Example 1 If the Checkerboard constructor is invoked and then the board is immediately drawn (i.e. no set methods are called on it), it creates a 3 column 2 row board with a height of 500 pixels and width of 750 pixels — see Figure 1 for how it should look. The method drawExample1 in CheckerboardComponent shows how the Checkerboard class is invoked. To make this example work you only need to implement the Checkerboard constructor and the drawOn method.

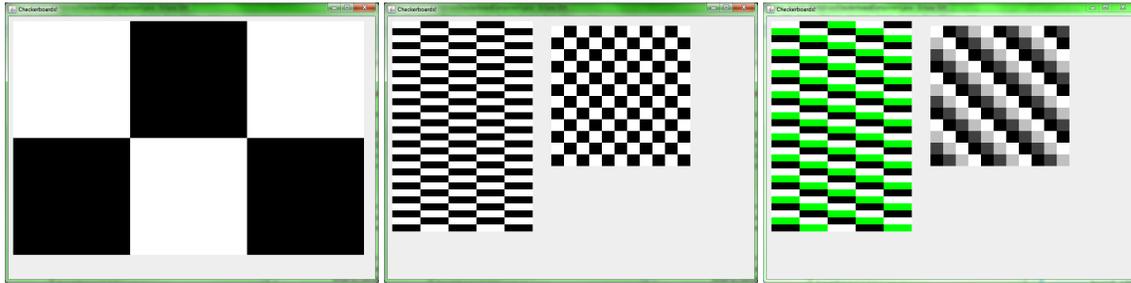


Figure 1: Example 1 (left). Example 2 (center). Example 3 (right).

Example 2 Now add functionality to set the height, width, number of columns, and number of rows. The method `drawExample2` shows how these functions are invoked — see Figure 1 for what it should look like when finished. You'll need to modify the `paintComponent` code in `CheckerboardComponent` very slightly to use `drawExample2`. To make this example work you need to implement `setHeight`, `setWidth`, `setNumberOfRows`, and `setNumberOfColumns`. You'll also likely need to update the way `drawOn` works.

Example 3 Now add the functionality to add new and different colors. Any arbitrary number of colors can be added by repeatedly calling the `addColor` function. Look at `drawExample3` to see how `addColor` is used (again you'll have to modify `paintComponent` to test using the `drawExample3`) — you can see the result in Figure 1. You'll need to implement `addColor` and modify the `drawOn` functionality to make this example work.

	Part A	Points	Earned
	<code>findLargest</code>	6	_____
	<code>multiplyImaginaryByReal</code>	6	_____
	<code>combineStrings</code>	6	_____
	<code>findFirstLetterMatches</code>	6	_____
	<code>count4s</code>	6	_____
	Part B Checkerboard		
	Example 1 functionality	10	_____
	Example 2 functionality	10	_____
	Example 3 functionality	10	_____
	Computer Part Subtotal	60	_____