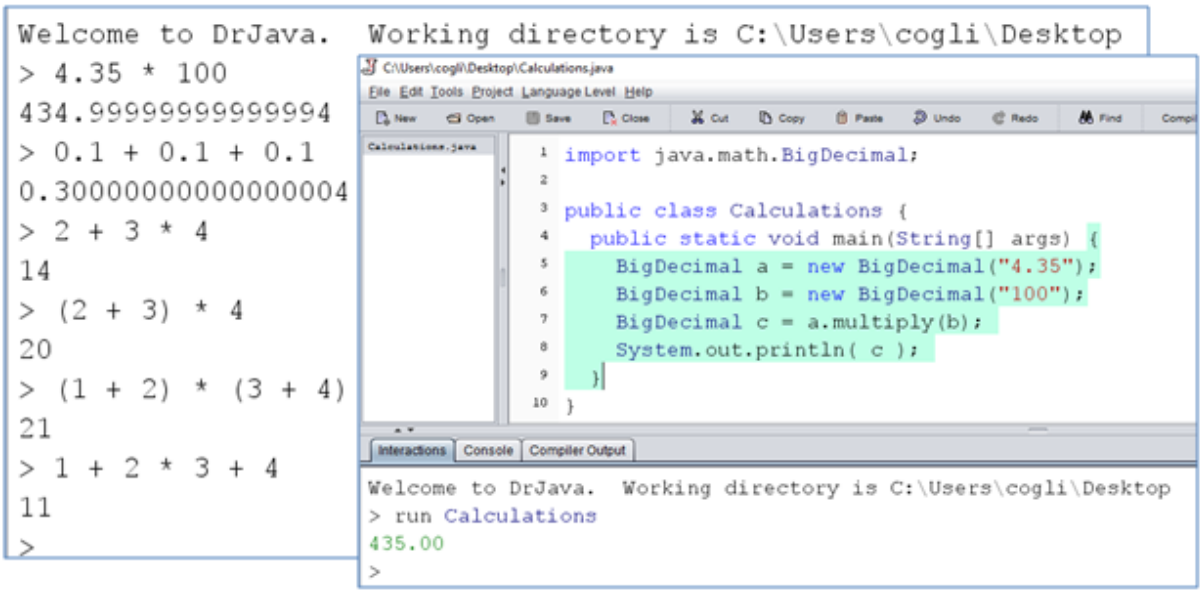
Please write your name and username here legibly: _____

C212/A592 ○ 6W2 Summer 2017 ○ Early Evaluation Exam: Fundamental Programming Structures in Java

Use `BigDecimal` (a class defined in package `java.math`) to write the following expressions in Java:

1. `4.35 * 100`
2. `0.1 + 0.1 + 0.1`
3. `2 + 3 * 4`
4. `(2 + 3) * 4`
5. `(1 + 2) * (3 + 4)`
6. `1 + 2 * 3 + 4`

Answer: Lab 01[1] is entirely identical to this group of six expressions[2]. This[3] is how we started the class on Monday June 19. Specifically Lecture Notes 01 directly address[4] the class, its purpose and syntax. So there can't be any misunderstanding whatsoever what this group of questions is asking. First, let's be sure we know why we do this, then let's calculate one expression completely:



Here's another way you can write the answer to the first question:

```
Welcome to DrJava.  Working directory is C:\Users\cogli\Desktop
> import java.math.BigDecimal;
> (new BigDecimal("4.35")).multiply(new BigDecimal("100"))
435.00
```

[1] https://www.cs.indiana.edu/classes/c212-dgerman/sum2017/lab01.html
[2] I graded this lab promptly from California and provided immediate feedback (see What's New? For Thu 06/22).
[3] https://www.cs.indiana.edu/classes/c212-dgerman/sum2017/0619a.html
[4] https://www.cs.indiana.edu/classes/c212-dgerman/sum2017/06-19-2017/image009.jpg

Having said this here's a perfectly good solution to the six questions as a group:

```
BigDecimal a = new BigDecimal("4.35");
BigDecimal b = new BigDecimal("100");
BigDecimal one = a.multiply(b);
System.out.println(one);
a = new BigDecimal("0.1");
BigDecimal two = a.add(a).add(a);
System.out.println(two);
a = new BigDecimal("1");
b = new BigDecimal("2");
BigDecimal c = new BigDecimal("3");
BigDecimal d = new BigDecimal("4");
BigDecimal three, four, five, six;
three = b.add(c.multiply(d));
System.out.println( three );
four = (b.add(c)).multiply(d);
System.out.println( four );
five = (a.add(b)).multiply(c.add(d));
System.out.println( five );
System.out.printf( "(%s + %s) * (%s + %s) = %s \n", a, b, c, d, five );
six = a.add(b.multiply(c)).add(d);
System.out.println( six );
```

You will notice we print the answer to the fifth problem twice to remind you of formatted printing.

Evaluate the following Java expressions:

7. `5 % 3`
8. `-5 % 3`

Answer: These two questions each aim to determine the remainder to an integer division. How many threes are there in five? There is only one three in five. If we take all the threes out (and we agreed there is only one) from five we're left with a remainder of 2 (two) which is the answer for question #7. As far as question #8 is concerned the correct answer to it is 1 (one) if you are a mathematician and -2 (negative two) if you are a Java programmer. To understand the difference[5] we remind you that classes of congruence modulo n only involve positive remainders whereas in Java remainders always have the sign of the dividend. Thus `-5 % 3 == -(5 % 3)` so the answer to question #8 is -2 (negative two).

```
Welcome to DrJava.  Working directory is C:\Users\cogli\Desktop
> 5 % 3
2
> -5 % 3
-2
```

9. `"substring".substring("length".length())`

Answer: This is the same as `"substring".substring(6)` which turns out to evaluate to `"ing"`.

[5] https://en.wikipedia.org/wiki/Modulo_operation#Common_pitfalls

10. `true || ! true && false`

11. `true || ! (true && false)`

12. `"This\nis\nnot\nit!".length()`

13. `"\\\\\\\\".length()`

14. `"mesquite in your cellar".replace('e', 'o')`

15. `"\\\\n\"".length()`

16. Write a Java `String` literal that prints as five backslashes: `\\\\\`

Answers. Exercises #9-16 can be easily tested as follows:

```
> true || ! true && false
true
> true || ! (true && false)
true
> "This\nis\nnot\nit!".length()
15
> "\\\\\\\\".length()
4
> "mesquite in your cellar".replace('e', 'o')
"mosquito in your collar"
> "\\\\n\"".length()
4
> "\\\\\\\\\\"
"\\\\\"
> System.out.println("\\\\\\\\\\")
\\\\\
```

Questions like question #14 are discussed in the book (see, for example, the self-check exercises on page 45). The book encourages us to check the `String` API and specifically the `replace` method and that's where you will find[6] this exact substitution example discussed. Questions #12, #13, #15, #16 simply check what we have learned and practiced[7] in Homework 01: special characters (like double quote, new line, backslash, etc.) need to be escaped in `String`s and in the process the representation becomes longer. However a special character is still one character even if we need two characters to represent it. Questions #10 and #11 are very basic and check your understanding of the order of precedence of the boolean operators. For example `!true && false` evaluates to `false` (since the negation operator binds closest and therefore acts first) while `!(true && false)` evaluates to `true` since the parens delay the logical negation until the very end.

[6] https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#replace-char-char-
[7] https://www.cs.indiana.edu/classes/c212-dgerman/sum2017/hw01.html

Simplify the following expressions where `b` is a `boolean` variable and `n` is an integer:

17. `b == true`

18. `b == false`

19. `b && !b`

20. `b || !b`

Answers: `b`, `!b`, `false`, and `true`. We discussed these in class (lecture and lab) via truth tables. If you want to convince someone (or yourself) that this is indeed the case you can write code to test your truth table. Take a look at the demonstration for question #18 below, in Java:

```
public class Simplifications {
  public static void main(String[] args) {
    boolean b;
    System.out.println(   " b      b == false    ! b  ");
    System.out.println(   "-------------------------------");
    b = false;
    System.out.println( b + "      " + (b == false) + "       " + (! b) );
    b = true;
    System.out.println( b + "      " + (b == false) + "       " + (! b) );
  }
}
```

Here's how this program runs (and prints the truth table):

```
Welcome to DrJava.  Working directory is C:\Users\cogli\Desktop
> run Simplifications
 b      b == false    ! b
--------------------------------
false      true       true
true      false       false
>
```

21. `n > 3 && n > 5`

22. `n > 3 && n < 5`

Answers: `n > 5` for question #21 and `n == 4` for question #22. To understand the thought process remember these are boolean expressions. We need to plot their truth tables to then try to guess what other function, with a more compact representation, would be able to behave the same over all `n`'s.

```
n                          3   5
-----------------------------------------
n > 3                F F F F F T T T T T T
-----------------------------------------
n > 5                F F F F F F F T T T T
-----------------------------------------
(n > 3) && (n > 5)   F F F F F F F T T T T
-----------------------------------------
```

You can see in this table that the same behavior is obtained with just `(n > 5)`. In a similar fashion we determine the answer to question #22: the points of interest are still 3 and 5, but the second term in the expression is different. So drawing the table again we obtain:

n			3		5					
n > 3	F	F	F	F	F	T	T	T	T	T
n < 5	T	T	T	T	T	T	F	F	F	F
(n > 3) && (n < 5)	F	F	F	F	F	T	F	F	F	F

So the steps (as we indicated in class last week) are:

(a) identify the variables (here: just n)

(b) identify the points of interest (here: 3 and 5)

(c) draw the truth table for the subexpressions

(d) aggregate that into the truth table for the entire expression

(e) think hard what simpler expression might match the resulting table

We proceed in the same fashion for the other expressions listed below:

Example:

23. `n < 3 && n > 5`

    `false`

24. `n < 3 || n > 5`

    Can't be simplified further (see below).

25. `n > 3 || n > 5`

    `n > 3`

26. `b && true`

    `b`

27. `b && false`

    `false`

28. `b || true`

    `true`

29. `b || false`

    `b`

Sometimes the answer to step (e) is: none. In that case it's still worthwhile to try to find equivalent expressions. We now need to discuss #24. Can we simplify `n < 3 || n > 5` in Java? Here's the table:

n	3				5					
n < 3	T	T	T	T	F	F	F	F	F	F	F
n > 5	F	F	F	F	F	F	F	T	T	T	T
n < 3 || n > 5	T	T	T	T	F	F	F	T	T	T	T
not 3 < n < 5 (python)	T	T	T	T	F	F	F	T	T	T	T
(not (< 3 n 5)) (racket)	T	T	T	T	F	F	F	T	T	T	T
?(n) (java)	T	T	T	T	F	F	F	T	T	T	T

So it seems we need to state (in Java) that n is not in the set {3, 4, 5}. Can we do that?

Can we simplify `24. n < 3 || n > 5` using Java syntax?

I say that the answer to this question is: no[8].

Can we write an equivalent expression (though not necessarily simpler)?

That's an entirely different proposition.

```
Can you write an equivalent expression (not necessarily simpler)?

Answer: definitely.

Examples:

  (n != 3) && (n != 4) && (n != 5) // 22 characters

  ! ("3:4:5".contains(n + "")) // 25 characters

  (n - 3) * (n - 4) * (n - 5) != 0 // 20 characters

  Math.abs(n - 3) > 1 // 15 characters
```

Sadly none of these is a simplification[9] (since the original expression is only 8 characters long).

Evaluate the following Java expressions:

30. `1 / 2 * 4`

    0 (zero)

31. `4 * 1 / 2`

    2

In the absence of parens evaluate `*` and `/` left to right. All operands are `int`s so be careful.

32. `(41 - 32) * 5 / 9`

    5

33. `5 / 9 * (41 - 32)`

    0

The first expression better approximates (in `int`s) the Celsius value of 41F.

34. If `a` and `b` are boolean variables is `!(a && b)` equivalent with `(!a || !b)`? Why or why not?

```
public class Simplifications {
  public static void main(String[] args) {
    boolean a, b;
    System.out.println(" a    b    !(a && b)   !a || !b  ");
    System.out.println("-----------------------------------");
    a = true; b = true;
    System.out.println(a + "  " + b + "  " + (!(a && b)) + "      " + (!a || !b));
    a = true; b = false;
    System.out.println(a + "  " + b + "  " + (!(a && b)) + "      " + (!a || !b));
    a = false; b = true;
    System.out.println(a + "  " + b + "  " + (!(a && b)) + "      " + (!a || !b));
    a = false; b = false;
    System.out.println(a + "  " + b + "  " + (!(a && b)) + "      " + (!a || !b));
  }
}
```

[8] And that should be a perfectly legitimate answer: not every expression can be simplified.

[9] But they're interesting equivalent expressions aren't they?

Here's what that program produces (a truth table):

```
Welcome to DrJava.  Working directory is C:\Users\cogli\Desktop
> run Simplifications
 a     b     !(a && b)    !a || !b
-------------------------------------
true   true   false       false
true   false  true        true
false  true   true        true
false  false  true        true
>
```

This proves the equivalence known as DeMorgan[10] law (one of them, there is a dual to this one).

35. If `m` and n are `int` variables and `n` is not zero can the following expression be simplified?

```
m / n * n + m % n
```

If not briefly explain why. If yes what is the value?

Answer: yes, `m`, since an integer division always produces a quotient (`m/n`) and a remainder (`m%n`).

36. What values are in n and m at the end of the following code fragment:

```
int n = 3, m = 5; n = m + n; m = n - m; n = n - m;
```

Answer: the values in `m` and `n` are switched so `n` is `5` and `m` is `3` at the end.

What are the types of each of the following Java expressions:

37. `Math.sqrt(2)`

`double`

38. `System.out`

`java.io.PrintStream`

39. `3` `'3'` and `"3"`

`int` `char` `String`

Clearly these are all expressions. Every expression has a value, every value has a type. The square root of a number (in this case 2) is a fractional number. In Java the default for that (as can be seen in the return type part[11] of the signature of the method invoked) is `double`. The textbook introduces the answer to question #38 as early as Chapter 1 (see page 24 at the top, end of chapter summary). You can also find the type I am asking for by just typing the expression in DrJava at the prompt (in the Interactions Panel):

```
Welcome to DrJava.  Working directory is C:\Users\cogli\Desktop
> System.out
java.io.PrintStream@1f850d4
>
```

As for the answers to #39 (a variation on question #7 from your text, p. 40) we already discussed in class that `3` is an `int` value (you need type `3l` if you want a `long` value instead), `3.0` is a double (`3.0f` is how you indicate that you want a `float`) an we also enumerated very clearly the differences between `char`s and `String`s and their respective delimiters `'` and `"` (and compared even with Python).

---

[10] See your textbook, page 213.

[11] https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html#sqrt-double-

40. What is wrong with the following loop for finding the position of the first space in a `String str`?

```
boolean found = false;
for (int position = 0;
     !found && position < str.length();
     position++){
  char c = str.charAt(position);
  if (c == ' ') {
    found = true;
  }
}
```

Answer: This is taken straight from your book[12] and because of that I just don't want to say anything about its answer here. I find that you are disrespecting your book so much (at least some of you) at this time that it mandatory for me to ask that you please check the answer in your text for this question.

41. What is wrong with the following loop for reading a sequence of values?

```
System.out.println("Enter values, Q to quit: ");
do {
  double value = in.nextDouble();        // in is a java.util.Scanner
  sum = sum + value;          // sum, count defined earlier as doubles
  count++;                  // both properly initialized upon creation
} while (in.hasNextDouble());
```

Answer: see your textbook, page 262, self-check question #25.

42. Suppose Java didn't have a do loop. Could you rewrite any do loop as a while loop? Explain w/ code.

Answer: see text page 258 self-check question #18.

Quick summary of answers:

```
1. import java.math.BigDecimal;
   // then somewhere in a method
   BigDecimal a = new BigDecimal("4.35"),
              b = new BigDecimal("100");
   BigDecimal one = a.multiply(b);
2. BigDecimal c = new BigDecimal("0.1");
   BigDecimal two = c.add(c).add(c);
3. BigDecimal d = new BigDecimal("2"),
              e = new BigDecimal("3"),
              f = new BigDecimal("4");
   BigDecimal three = d.add(e.multiply(f));
4. BigDecimal g = new BigDecimal("1");
```

[12] Page 271 in your text self-check question 34 (continues on page 272).

```
    BigDecimal four = (d.add(e)).multiply(f);
 5. BigDecimal five = (g.add(d)).multiply(e.add(f));
 6. BigDecimal six = g.add(d.multiply(e)).add(f);
 7. 2
 8. -2
 9. "ing"
10. true
11. true
12. 15
13. 4
14. "mosquito in your collar"
15. 4
16. "\\\\\\\\\\"
17. b
18. !b
19. false
20. true
21. n > 5
22. n == 4
23. false
24. Can't be simplified further.
25. n > 3
26. b
27. false
28. true
29. b
30. 0
31. 2
32. 5
33. 0
34. Yes, truth table. DeMorgan's law.
35. Yes, it is equal to m. Simple arithmetic (quotient, remainder).
36. Values are swapped between n and m.
37. double
38. java.io.PrintStream
39. int, char and String
40. Code runs well but position is not available after the loop ends.
41. If we start by typing a non-number the code crashes.
42. Yes, like this:

do {                        boolean first = true;
  <body>;                   while (first || <cond> ) {
} while ( <cond> );           <body>;
                              first = false;
                            }
```