In-Class + Home Assignment: Expressions and Evaluations

5 points for completing all; 2 for partial completion; 0 if not submitted

In class, we talked about a lot of different expressions that you could write in your program.

We talked about the order in which operators get evaluated. The computer evaluates operations in the following order:

1. Operations inside parentheses
2. Multiplication operations and Division operations
3. Addition and Subtraction operations

We also talked about augmented notations (or shorthand notations) such as +=, -=, *=, and /=. We also talked about increment (++) and decrement operators (--).

We also talked about typecasting, that is, changing data types from one type to another. We talked about how the computer internally converts data type when it is compatible and how we can do it explicitly (e.g., by writing `(int) 2.3` we are converting a double type value 2.3 to an integer value).

In this exercise, you are going to practice using these operators.

Step 1: Fill in the ***first*** blank column

First, you are going to **fill in the second column only**. Think what the expression will be evaluated to and write them down. Some of them have errors. In those, write down “Error” and briefly share why you think it is an error.

Addition operator:

Expression	**Value of x (my guess/evaluation)**	**Output from the program**
int x = 2 + 3;	5	5
int x = 2.0 + 3.0;	5	error
double x = 2 + 3;	5.0	5.0
double x = 2.0 + 3;	5.0	5.0
boolean x = 2.0 + 3.0;	true	error

Subtraction operator:

Expression	Value of x (my evaluation)	Value of x (computer's evaluation)
int x = 2 - 3;	-1	-1
int x = 2.0 - 3.0;	-1	Error (Lossy conversion from double to int)
double x = 2 - 3;	-1.0	-1.0
double x = 2.0 - 3;	-1.0	-1.0
boolean x = 0 - 1;	false	Error (Double cannot be converted to boolean)

Multiplication operator:

Expression	Value of x (my evaluation)	Value of x (computer's evaluation)
int x = 2 * 3;	6	6
int x = 2.0 * 3.0;	6	Error (Double cannot be converted to int)
double x = 2 * 3;	6.0	6.0
double x = 2.0 * 3.0;	6.0	6.0

Division operator:

Expression	Value of x (my evaluation)	Value of x (computer's evaluation)
int x = 13 / 0;	error	Error (division by zero)
int x = 13 / 4;	3	3
int x = 13 / 4;	3	3
int x = 13.0 / 4;	3	Error (Cannot convert double to int)
double x = 13 / 4;	3.25	3.0 (Integer division floors the actual answer, so 3.25 becomes 3.0)
double x = 13.0 / 4.0;	3.25	3.25
double x = 13 / 4;	3.25	3.0 (Integer division floors actual value of 3.25)
double x = 13**.** / 4;	3.25	3.25

Modulo (or remainder) operator:

Expression	Value of x (my evaluation)	Value of x (computer's evaluation)
int x = 12 % 0;	error	Error (division by zero)
int x = 12 % 4;	0	0
int x = 14 % 4;	2	2
int x = 4 % 14;	4	4
double x = 13 % 4;	1.0	1.0
double x = 13.0 % 4.0;	1.0	1.0

Operator precedence:

Expression	Value of x (my evaluation)	Value of x (computer's evaluation)
int x = 2 + 3 * 5;	17	17
int x = (2 + 3) * 5;	25	25
double x = (2 + 3) – 6 / 2;	2.0	2.0
double x = ((2 + 3) – 6) / 2;	-1.0	0 (Integer division floors the new value)
double x = (2 + 3) * 5 – 6.0 / 2;	22.0	22.0
Double x = 2 + 3 * 5 – 6.0 / 2;	14.0	14.0

Augmented notations: Mark the outcome after ALL statements have been executed.

Expression	Value of x (my evaluation)	Value of x (computer's evaluation)

int x = 3;
x++;
	4	4

int x = 3;
x += 2;
	5	5

double x = 2.3; x--;	1.3	1.2999999999999998

double x = 2.3;
x -= 1;
	1.3	1.2999999999999998

int a = 5;	54	50 (It seems that post

int b = 10; int x = a++ * b--;		increment increments a and b after it has assigned the value to x)

int a = 5;
int b = 10;
int x = ++a * b--;
	54	60 (The pre-increment adds 1 to a before it is multiplied, b is then decremented after x is assigned the value of 60)

[More Content Below]

Typecasting (changing data types): Mark the outcome after ALL statements have been executed.

Expression	**Value of x (my evaluation)**	**Value of x (computer's evaluation)**

int y = 3;
double x = y;
	3.0	3.0

int y = 3;
double x = (double) y;
	3.0	3.0

double y = 3.0;
int x = y;
	3	Error (lossy conversion)

double y = 3.0;
int x = (int) y;
	3	3

double x = 2.3;
x -= 1;
x = (int) x;
	1.0	1.0

Step 2: Fill in the ***second*** blank column

Now, create a method in Java and write the statements one by one and see what output you get. You can use the code from the box below.

```
    public static void main(String[] args) {
        int x = 3 + 4; // replace this with your expression

        System.out.println(x);
    }
}
```

Write down the results on the side and see which matched and which didn’t. For those that didn’t match with your expected outcome, think why it didn’t match.

My thoughts on the outcomes:
One thing that I found interesting with the results is that Java actually throws an error when there is potential data loss (e.g. conversion from double to int without the use of casting). In C, if you forget to cast the variables, it would just throw a warning and would truncate the new value.

Another thing is the behavior of post-increment/decrement and pre-increment/decrement operators. Post-increment/decrement operators change the value of the operand after the assignment, while pre-increment/decrement changes the value of the operand before the assignment happens.

[More Content Below]

Surprising Results:

There may be some outcome that surprised you. Please note some of those expressions below and we will discuss them in the next class.

Expression	What you expected	What you got

int a = 5;
int b = 10;
int x = a++ * b--;
	54	50

int a = 5;
int b = 10;
int x = ++a * b--;
	54	60
