Lab 3: Expressions and Variables

Credit to Stuart Reges and Marty Stepp of the University of Washington for the contents of this content. This lab document was created by Marty Stepp, Stuart Reges and Whitaker Brand

Goals for Lab

- Write and evaluate expressions to compute numeric values
- Use variables to store results of a computation into memory

Operators

Java has operators for the standard mathematical operations. Note that + works a little differently depending on the type of the operands.

Operator	Description
+	Addition for ints and doubles, concatenation for Strings
-	Subtraction for ints and doubles, doesn't compile for Strings
*	Multiplication for ints and doubles, doesn't compile for Strings
/	Division for ints and doubles, doesn't compile for Strings. Note that ints will follow integer (truncated) division rules
%	Modulus for ints and doubles, doesn't compile for Strings

Precendence

Generally speaking, Java tries to evaluate your expression from left to right. However, some operations happen before others. Java follows a set of rules of precedence that should be familiar from Algebra class, abbreviated PEMDAS.

Java doesn't have an exponent operator, but it does have the modulus (%) operator, so the mnemonic becomes:

PMMDAS

- P :: first, evaluate expressions surrounded by Parentheses ()
- MMD :: then, perform all Multiplication *, Modulus %, and Division / operations
- AS :: finally, go through and perform the Addition +, and Subtraction -

Expressions

Recall that Java has expressions to represent math and other computations. Expressions may use operators, which are evaluated according to rules of precedence. Every expression produces a value of a given type.

Type	Description	Expression Example	Result

Type	Description	Expression Example	Result
int	integers (up to 231 - 1)	3 + 4 * 5	23
double	real numbers (up to 10308)	3.0 / 2.0 + 4.1	5.6
String	text characters	"hi" + (1 + 1) + "u"	"hi2u"

Exercise 1: Expressions

Write the results of each of the following expressions. If you're stuck, ask a neighbor or your teacher.

12 / 5

12.0 / 5

12 / 5 + 8 / 4

3 * 4 + 15 / 2

42 % 5 + 3 % 16***

***Hint: Suppose you have 3 apples, and you distribute them among 16 people so that everyone gets the same number of (whole) apples. How many (whole) apples does each person get? How many apples do you have left over?

String Concatenation

The + operator does addition on numbers, but on Strings, it instead appends the two Strings together.

```
    System.out.println("Hello, World!");         // prints: Hello, World!
    System.out.println("Hello, " + "World!");    // prints: Hello, World!

    System.out.println("You " + "can" + " add" + "many" +   "   Strings together");
 // prints: You can addmany   Strings together
    System.out.println("Numbers " + 2 + "!");  // prints: Numbers 2!
```

Note that only the whitespace inside the quotation marks is preserved.

Using quotation marks tells Java to keep track of every character from the opening " to the closing ", including the whitespaces like spaces, tabs, and newlines.

Outside the quotes, we're back in the Java program itself, where Java doesn't notice or care about the newline we've used to format the long line of source code.

String Concatenation with Numbers

We do String concatenation instead of addition if either of the operands are Strings.

We only add the numbers together if both of them are numbers (not Strings). Java keeps track of the type of pieces of data in your program. It treats "4" (a String) differently than it treats 4 (a numerical value).

Having the distinction is crucial for some tasks: if the computer always treated numerical Strings as numbers, then we wouldn't have a way to prepend an area code to the beginning of a phone number.

```
// All are using the + on a String -- all perform concatenation.
System.out.println("1" + "2");      // prints 12
System.out.println("1" + 2);        // same, prints 12
System.out.println(1 + "2");        // same, prints 12

// Since both operators here are numbers, we do addition:
System.out.println(1 + 2);     // does addition before printing -- prints 3
```

Note: Even though the + operator works differently on Strings, it still follows the same precedence rules.

Exercise 2: More expressions

Write the results of each of the following expressions.

Some of these expressions include String concatenation. When the result of the expression evaluates to a String, put "quotation marks" around the result:

`"x" + 2` // example answer: "x2"

`2 + 6 + "cse 142"`

`"ap cs" + 2 + 6`

`1 + 9 / 2 * 2.0`

`46 / 3 / 2.0 / 3 * 4/5`

jGRASP interactions pane

jGRASP has a useful feature called the Interactions Pane that allows you to type in Java expressions or statements one at a time and instantly see their results.

To use it, run jGRASP and then click the "Interactions" tab near the bottom.

Exercise 3: Using jGrasp Interactions Pane

In this exercise, you'll use the Interactions Pane to quickly discover the result of some expressions that would be difficult to evaluate by hand. Copy/paste each expression below into the Interactions Pane to evaluate it, then input the answer into this slide.

`123 * 456 - 789`

`3.14 + 1.59 * 2.65`

`2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2`

`2 + 2 + "xyz" + 3 + 3`

Variables

Recall that you can use a variable to store the results of an expression in memory and use them later in the program.

```
type name;                          // declare
name = value or expression;         // assign a value
...
type name = value or expression;    // declare-and-initialize together
```

Examples:

```
double iPhonePrice;
iPhonePrice = 299.95;

int siblings = 3;
System.out.println("I have " + siblings + " brothers/sisters.");
```

Exercise 4: Variable declaration syntax

Which of the following choices is the correct syntax for declaring a real number variable named grade and initializing its value to 4.0?

- [ ] double grade = 4.0;
- [ ] grade = 4.0;
- [ ] grade : 4.0;
- [ ] int grade = 4.0;
- [ ] 4.0 = grade;
- [ ] grade = double 4.0;

Exercise 5: Variable Assignment syntax

Suppose you have a variable named grade, set to 1.6:

```
double grade = 1.6;   // uh-oh
```

Suppose later in the program's code, we want to change the value of grade to 4.0. Which is the correct syntax to do this?

- [ ] set grade = 4.0;
- [ ] grade = 4.0;

- [ ] double grade = 4.0;
- [ ] 4.0 = grade;
- [ ] grade : 4.0;

Exercise 6: Variable Mutation

Suppose you have a variable named balance, set to 463.23:

```
double balance = 463.23
```

Suppose later in the program's code, we want to add 5 to the account balance. Which is a correct statement to do this?

- [ ] balance = balance + 5;
- [ ] balance = 5;
- [ ] balance + 5;
- [ ] 5 + balance = balance;
- [ ] balance <-- 5;

Exercise 7: a, b, c

What are the values of a, b, and c after the following statements? Write your answers in the boxes on the right.

```
int a = 5;
int b = 10;
int c = b;
```

a = a + 1; // a? [____________]

b = b - 1; // b? [____________]

c = c + a; // c? [____________]

Exercise 8: i, j, k

What are the values of i, j, and k after the following statements?

int i = 2; int j = 3; int k = 4; int x = i + j + k;

i = x - i - j; // i? [____________]

j = x - j - k; // j? [____________]

k = x - i - k; // k? [___________]

Exercise 9: Using the Debugger

jGRASP has a debugging tool that you can use to help identify errors or bugs in your code. Create a new Java file in jGRASP and copy and paste the code below into that file.

```
public class Grader {
  public static void main(String[] args) {
     // pa1=10, pa2=16, pa3-8 = 20, section=20
     int homeworkTotal = 10 + 16 + 6 *20 + 20;
     int homeworkScore = 8 + 14 + 6 * 18 + 20;
     int extraCredit = 3;
     double homework = 100.0 * (homeworkScore + extraCredit / 3) / homeworkTotal;
     System.out.println("overall = " + homework);
  }
}
```

The purpose of this program is to calculate the grade of a student. In this version, we have added extra credit to the student's homework score. Specifically, extra credit is calculated as total extra credit points divided by 3.

Compile and run your code. You should see the following output on the console:

```
 ----jGRASP exec: java Grader
overall = 90.96385542168674

 ----jGRASP: operation complete.
```

Now, change the variable `extraCredit` to 4. Compile and run your code. You should see the following output on the console:

```
 ----jGRASP exec: java Grader
overall = 90.96385542168674

 ----jGRASP: operation complete.
```

It appears that the score hasn't changed even though the student did more extra credit. Try running the program with extra credit scores of 5, 6, 7, 8, 9. What do you notice about the scores? Shouldn't we expect a student's overall score to change as they rack up more and more extra credit?

Obviously this program has a bug. To identify the bug, let's use the debugger and the interactions pane. Do the following:

- Using your mouse pad, hover just to the left of the number `4` on line 4 until you see a red dot. Click the mouse to set the red dot as our break point.
- Click the LadyBug icon to start the debugging.
- Step through the debugger by clicking the downward pointing blue arrow, right underneath the floppy disk icon on the top left of the jGRASP editor.
- As you step through, note what the values are for each variable.

It appears that line 7 is being calculated incorrectly. Use the interactions pane to run the mathematical logic (i.e. the grading algorithm). Break the algorithm into separate pieces to help identify what the bug is. When you've resolved the bug, fix the program and paste you updated code below.