

Unit 1: Primitive Types

Topic 3: Expressions & Assignment Statements

Submit in Google Classroom when complete

Name: _____

PREDICT: Without Replit, mentally predict what this code segment will print; type your predictions in the column on the right.

```
int x = 5;
int y = 10;
System.out.println("x: " + x + ", y: " + y);

x = y;
System.out.println("x: " + x + ", y: " + y);

y = 20;
System.out.println("x: " + x + ", y: " + y);

int z = x + y;
System.out.println("x: " + x + ", y: " + y);
System.out.println("z: " + z);

x = 7;
System.out.println("x: " + x + ", y: " + y);
System.out.println("z: " + z);
```

PREDICT (first one done)

x: 5, y: 10

NOW TEST! Once you have typed your predictions, copy/paste the code segment above into Replit and run it.

[Miss any? See the explanations](#)

Challenge!

Below is a code segment. **Add some code** in the `// add code here` section to **swap** the values in `num1` and `num2` so that when you execute the code, the printed output is:

```
num1 = 10 num2 = 15
num1 = 15 num2 = 10
```

You *could* simply assign 15 to `num1` and 10 to `num2`, but that's too easy -- can you figure out a way to do this using assignment statements (`=`) that involve *only* `num1`, `num2`, and a *third*, new variable named `temp`?

```
int num1 = 10;
int num2 = 15;
System.out.println("num1 = " + num1 + " num2 = " + num2);
// add some code here!
System.out.println("num1 = " + num1 + " num2 = " + num2);
```

[hint!](#)

When you have it figured out, compare with your partner! Did you do it the same way?

Copy/paste your solution below:

[give up? \(don't give up!\)](#)

Operator precedence is the same as math: **PEMDAS** with modulo added with mult and div:

first ()

then * / % (L to R)

lastly + - (L to R)

(we will talk about exponents later!)

PREDICT: Without Replit, **mentally predict** what this code segment will print; type your predictions in the column on the right.

```
int a = 3;
int b = 4;
int c = 5;

a = b + 2 * c - 10 / 5;
System.out.println("a = " + a);

a = (b + 2) * (c - 10) / 5;
System.out.println("a = " + a);
```

PREDICT:

a =

a =

NOW TEST! Once you have typed your predictions, copy/paste the code segment above into Replit and run it.

[Miss any? Explanations](#)

EXPLORE: Copy/paste and run the code below to see what happens with **division by 0** (You can delete the code from the problem above involving a, b, and c)

```
int a = 3;
int b = 4;
System.out.println(a);
System.out.println(b);

int c = (5 * b) / (a - 3);
System.out.println(c);
```

What error message do you see?

This is called an **ArithmeticException** -- an **exception** is a **runtime error** which results from a problem in your code that occurs *while the program is actually running!*

Important!

How can you tell the program is actually running before this runtime error/exception occurs?

How is a *runtime* error **different** from a *compile-time/syntax* error?

After you have thought about these, [check your answers here!](#)

Quick mod review!

Mentally predict what this code segment will print; type your predictions in the column on the right.

```
int e = 20 % 5;  
int f = 19 % 5;  
int g = 18 % 5;  
int h = 17 % 5;  
int i = 7 % 5;  
int j = 5 % 5;  
int k = 3 % 5;  
int l = 1 % 5;
```

```
System.out.println("e = " + e);  
System.out.println("f = " + f);  
System.out.println("g = " + g);  
System.out.println("h = " + h);  
System.out.println("i = " + i);  
System.out.println("j = " + j);  
System.out.println("k = " + k);  
System.out.println("l = " + l);
```

PREDICT:

NOW TEST! Once you have typed your predictions, copy/paste the code segment above into Replit and run it.

[See the explanations](#)

Math expressions involving **two INTS** will result in an **INT**
Math expressions involving **two DOUBLES** will result in a **DOUBLE**
Math expressions involving an **INT** and a **DOUBLE** will result in a **DOUBLE**

Mentally predict what this code segment will print; type your predictions in the column on the right.

```
System.out.println(3 + 2);  
System.out.println(3 + 2.0);  
System.out.println(3.0 + 2);  
System.out.println(3.0 + 2.0);
```

PREDICT:

NOW TEST! Once you have typed your predictions, copy/paste the code segment above into Replit and run it.

[See explanation](#)

Math involving **DIVISION** is tricky! Dividing **two INTS** will result in an **INT**, but the result is **TRUNCATED** (not rounded; the decimal is literally chopped off).

This is known as "int division" and is super important!

If you don't want to "lose the decimal", make sure at least one value being divided is a **double**.

Mentally predict what this code segment will print; type your predictions in the column on the right.

```
System.out.println(10 / 5);  
System.out.println(10 / 4);  
System.out.println(10 / 3);  
System.out.println(3 / 2);  
System.out.println(1 / 3);  
System.out.println(2 / 3);  
System.out.println(9 / 10);  
  
System.out.println(10.0 / 5);  
System.out.println(10.0 / 4.0);  
System.out.println(10 / 3.0);  
System.out.println(3.0 / 2.0);  
System.out.println(1.0 / 3);  
System.out.println(2 / 3.0);  
System.out.println(9.0 / 10.0);
```

PREDICT:

NOW TEST! Once you have typed your predictions, copy/paste the code segment above into Replit and run it (or try to!)

[Miss any? Be sure to read the explanations!](#)

Tyler wrote the following code to calculate and print his test average as a percent on a test he took worth 60 points (he scored 55 out of 60).

```
int earnedPoints = 55;
int totalPoints = 60;
double testPercent = earnedPoints / totalPoints * 100;
System.out.println("Test average: " + testPercent);
```

PREDICT: Will this correctly print his test average as a percent? If not, what do you think is the problem?

NOW TEST! Once you have typed your predictions, copy/paste the code segment above into Replit and run it! Does it print the correct percent as expected?

Debugging: Help Tyler fix the mistake! Make a change to the code so that the correct average (91.66..) prints (there are a couple things you could change!)

Paste the *fixed* code below:

[check](#)

Tyler *initially* tried to make the 100 a double, like shown below, but discovered that it didn't work (try it if you don't believe it!):

```
int earnedPoints = 55;
int totalPoints = 60;
double testPercent = earnedPoints / totalPoints * 100.0;
System.out.println("Test average: " + testPercent);
```

What was the error in Tyler's initial attempt to fix the program himself? (in other words, why doesn't the code above work as expected)?

[check](#)

Tyler was on to something though! Leaving both earnedPoints and totalPoints as **ints**, move the 100.0 *somewhere else* so that it **DOES** work!

Paste your code below:

[check](#)

PREDICT: Mentally predict if this code will **compile** -- in other words, if you copy/paste it, there will be no red squiggles in Replit! (don't copy/paste yet, think with your brains first)

```
double s = 8.25;
int t = 2;
int u = 10;

double m = t + u;
int d = s + t;
double v = s / (t + 0.75);
int w = 2 * s;
```

Write your prediction: Will the code compile as written (assume it appears inside a main method in the Main class)? If not, why not?

NOW TEST! Once you have typed your predictions, copy/paste the code segment above into Replit and run it (or try to!)

[See answer and explanation](#)

Was your prediction correct? If not, why not?

PREDICT: *Mentally* predict what these two code segments will print; type your predictions in the column on the right.

```
int x = 5;
int y = 9;
System.out.println(4 + x * 6 % y);
System.out.println(y % x / 5);
```

PREDICT:

```
int p = 3 / (5 % 3);
int q = (2 + 4) * 3;
double r = 2 * p + q;
System.out.println("p = " + p);
System.out.println("q = " + q);
System.out.println("r = " + r);
```

PREDICT:

NOW TEST! Once you have typed your predictions, copy/paste each code segment above into the main method of your Workspace program and run it.

[Miss any? See the explanations](#)

Was your prediction correct? If not, why not?

Why does `r` print as **20.0** rather than **20**?

[Check](#)

Sample Program

A. [Open up this sample Java program](#) that prints out dimensional information about a rectangular prism and area. Study how it works! Pay careful attention to how variables are used, how the code is structured, and the use of comments. You will be building a similar program next

□

Note: `double` values are sometimes a little *imprecise* when printed out (such as 11.3333333334); this is due to computers not always managing decimal values very well, and we will learn some strategies for printing decimals nicely.

```
*****
My rectangular prism has these dimensions:
Length: 9
Width: 10
Height: 15
Volume: 1350
Average side length: 11.333333333333334
*****
My circle has radius:4.5
Area: 63.6171975
*****
```

Using your U1T3 Lab Replit, here is the program you should write:

Stacey went grocery shopping for ingredients to make a fruit salad with \$40 in cash and purchased the following:

- A dozen apples costing \$0.60 each.
- Two pineapples costing \$3.59 each.
- Three bags of rice, which costs \$1.39 *per pound* (each bag of rice weighs 2 kilograms)

Write a Java program that prints a “receipt” for Stacey’s grocery trip, showing:

- customer’s name
- how many items she purchased
- how much she spent in total
- the average cost per item
- the amount tendered (i.e. the amount she gave to the cashier)
- her change

Be sure to print the dollar signs too! □

Use comments to describe what “sections” of your program do

EXAMPLE: Below is an **example of the receipt** that Stacey received from her grocery trip *last* week, when she gave the clerk **\$5** to purchase **two** apples, **one** pineapple, and no bags of rice:

```
-----  
Customer: Stacey  
Number of Items: 3  
Total Cost: $4.79  
Average Cost: $1.596666666666667 per item  
  
Cash Tendered: $5.0  
Change Received: $0.209999999999999  
-----
```

NOTES:

- Like in the example above, you will likely have `double` values printed to the screen with various decimal lengths that don't look like dollars and cents, such as in the average cost per item and tendered -- don't worry about this for now, we will learn how to fix this!
- Please use *comments* in your code to help the reader of your code
- Please do *any and all conversions and calculations as part of your program* (i.e. don't do calculations off to the side and type in those results).
- Use variables in your program to represent *all* amounts.
- There is at least one place where a **constant** makes sense (hint: it has to do with conversions...)
- Have fun ☐!

When you are finished, copy/paste your full program code below and use the **Courier New** font.

Insert a screenshot of the printed receipt (console output):

Bonus Challenge! (Optional)

Although we haven't learned a strategy to do this yet, can you come up with a way to print out dollars and cents correctly for all monetary variables? For example, instead of \$5.0, print \$5.00, or instead of \$1.5966666666666667, print \$1.60. You might need to do a little research on "rounding in Java" or "string formatting" (two different approaches).

If you figure it out, **add it to your submitted program above, highlight the line(s) of code where you accomplish this**, then insert a **screenshot** of the *updated* receipt (with rounding) below:

Done!

Submit in Google Classroom:

Turn in

Answers ([back](#))

Below is the output you should be getting:

```
int x = 5;
int y = 9;
System.out.println(4 + x * 6 % y);
System.out.println(y % x / 5);
```

```
7
0
```

```
4 + x * 6 % y
= 4 + 5 * 6 % 9
= 4 + 30 % 9   (* / % from L to R)
= 4 + 3
= 7

y % x / 5
= 9 % 5 / 5
= 4 / 5
= 0   (because this is int division;
       0.8 truncates to 0)
```

```
int p = 3 / (5 % 3);
int q = (2 + 4) * 3;
double r = 2 * p + q;
System.out.println("p = " + p);
System.out.println("q = " + q);
System.out.println("r = " + r);
```

```
p = 1
q = 18
r = 20.0
```

```
int p = 3 / (5 % 3)
      = 3 / 2   (parentheses first)
      = 1       (int division truncates;
                  1.5 truncates to 1)

int q = (2 + 4) * 3
      = 6 * 3   (parentheses first)
      = 18

double r = 2 * p + q
         = 2 * 1 + 18
         = 2 + 18 = 20.0 (as double)
```

Answer ([back](#))

Why does r print as 20.0 rather than 20?

Because r is declared as a double: `double r`

So even though p and q are both `ints`, Java auto-converts the `int` value behind the scenes from 20 to 20.0 (a double)

Explanations ([back](#))

assign 5 to x and 10 to y

```
int x = 5;  
int y = 10;
```

prints out values stored in x and y, which are 5 and 10

```
System.out.println("x: " + x + ", y: " + y);
```

assign whatever value is in y to x, so this assigns 10 to x -- x doesn't know it got 10 from y, it just knows that it's 10

```
x = y;
```

prints out values stored in x and y; x now has 10 stored in it, and y still has 10 stored in it

```
System.out.println("x: " + x + ", y: " + y);
```

change the value stored in y to 20 -- note that we are changing y, but this has no effect on x; x is still 10 since it not "connected" in any way to y

```
y = 20;
```

prints out values stored in x and y; x still has 10 stored in it, and y now has 20 stored in it

```
System.out.println("x: " + x + ", y: " + y);
```

store the sum of whatever values are stored in x and y into z; x has 10, y has 20, so we store 30 into z -- z doesn't know it got 30 from x and y, it just knows that it's 30

```
int z = x + y;
```

prints out values stored in x, y, and z, which are 10, 20, and 30

```
System.out.println("x: " + x + ", y: " + y);
```

```
System.out.println("z: " + z);
```

change the value stored in x to 7-- note that we are changing x, but this has no effect on z; z is still 30 since it not "connected" in any way to x or y

```
x = 7;
```

prints out values stored in x, y, and z, which are 7, 20 (y is unchanged), and 30 (z is unchanged)

```
System.out.println("x: " + x + ", y: " + y);
```

```
System.out.println("z: " + z);
```

ACTUAL OUTPUT:

x: 5, y: 10

x: 10, y: 10

x: 10, y: 20

x: 10, y: 20
z: 30

x: 7, y: 20
z: 30

Hint ([back](#))

The "swap" of two variable values is a classic challenge, and can be accomplished by using a third "temporary" variable (temp). Assign either num1 (or num2) to temp first, then update num1 (or num2)... can you figure out the rest?

Solution ([back](#))

Here is one way to do it:

```
int num1 = 10;  
int num2 = 15;  
System.out.println("num1 = " + num1 + " num2 = " + num2);
```

```
// ADDED CODE:
```

```
int temp = num1; // assign num1 to a newly declared variable temp  
num1 = num2;     // assign num2 to num1  
num2 = temp;     // assign temp to num2
```

```
System.out.println("num1 = " + num1 + " num2 = " + num2);
```

Explanations ([back](#))

```
int a = 3;
int b = 4;
int c = 5;

a = b + 2 * c - 10 / 5;
a = 4 + 2 * 5 - 10 / 5; // substitute
a = 4 + 10 - 10 / 5; // mult first
a = 4 + 10 - 2; // div next
a = 14 - 2; // add next
a = 12 // subtract last
System.out.println("a = " + a);
```

ACTUAL OUTPUT:

a = 12

```
a = (4 + 2) * (c - 10) / 5;
a = (4 + 2) * (5 - 10) / 5; // substitute
a = (6) * (-5) / 5; // parentheses
a = -30 / 5; // mult
a = -6 // div
System.out.println("a = " + a);
```

a = -6

Important!

How can you tell the program is actually running before this runtime error/exception occurs?

How is a *runtime* error **different** from a *compile-time/syntax* error?

You can tell because these lines of code:

```
int a = 3;
int b = 4;
System.out.println(a);
System.out.println(b);
```

which are **before** where the division by 0 happens actually produce printed output:

```
3
4
```

And **then** the division by 0 occurs and the runtime error (exception) happens:

```
3
4
Exception in thread "main" java.lang.ArithmeticException
/ by zero
    at Main.main(Main.java:49)
exit status 1
```

Because the "3" and "4" both got printed, we know the program is running!

This type of error is different from a compile-time/syntax error, which prevents the program from even starting to run, and you need to fix the red squiggles before your program runs!

Runtime errors don't get caught by the Java compiler -- the syntax looks fine (i.e. there are no syntax "typos"), and the error only shows itself *after* the program begins to run.

Explanations ([back](#))

```
int e = 20 % 5;  
int f = 19 % 5;  
int g = 18 % 5;  
int h = 17 % 5;  
int i = 7 % 5;  
int j = 5 % 5;  
int k = 3 % 5;  
int l = 1 % 5;
```

```
System.out.println("e = " + e);  
System.out.println("f = " + f);  
System.out.println("g = " + g);  
System.out.println("h = " + h);  
System.out.println("i = " + i);  
System.out.println("j = " + j);  
System.out.println("k = " + k);  
System.out.println("l = " + l);
```

```
20 div 5 = 4 remainder 0  
19 div 5 = 3 remainder 4  
18 div 5 = 3 remainder 3  
17 div 5 = 3 remainder 2  
7 div 5 = 1 remainder 2  
5 div 5 = 1 remainder 0  
3 div 5 = 0 remainder 3  
1 div 5 = 0 remainder 1
```

ACTUAL OUTPUT:

```
e = 0  
f = 4  
g = 3  
h = 2  
i = 2  
j = 0  
k = 3  
l = 1
```

Explanation ([back](#))

<pre>System.out.println(3 + 2); System.out.println(3 + 2.0); System.out.println(3.0 + 2); System.out.println(3.0 + 2.0);</pre>	OUTPUT : 5 5.0 5.0 5.0
--	---

Only the $3 + 2$ prints as 5 (rather than 5.0); since 3 and 2 are both ints, the result is an int. The rest involve *at least one double*, so the result is a double and gets printed as 5.0 rather than 5.

Explanation ([back](#))

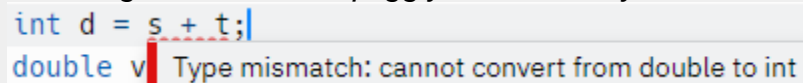
The code does **not compile**! There are few **compile-time/syntax** errors, outlined below:

```
double s = 8.25;
int t = 2;
int u = 10;
```

```
// this IS fine; t + u results in an int,
// but an int CAN be stored in a double!
double m = t + u;
```

```
// this is NOT fine; s + t results in a double, because s is a
// double (and a math operation between an int and double results
// in a DOUBLE), and we CANNOT store a double in an int variable!
int d = s + t;
```

mousing over the red squiggly shows the syntax error:

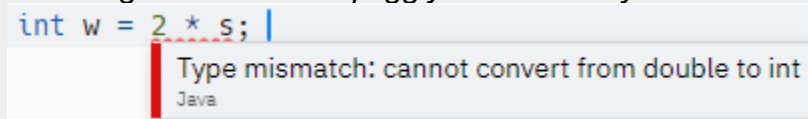


```
int d = s + t;
double v
```

```
// this is also fine; math involving s and t result in a double,
// and we are storing the result in a double variable
double v = s / (t + 0.75);
```

```
// this is NOT fine; again, math involving s will result in a
// double, because s is a double, and we CANNOT store a double in
// an int variable!
int w = 2 * s;
```

mousing over the red squiggly shows the syntax error:



```
int w = 2 * s;
```

Explanation ([back](#))

these are all int/int: this is int division and always results in truncation (not rounding)

```
System.out.println(10 / 5);  
System.out.println(10 / 4);  
System.out.println(10 / 3);  
System.out.println(3 / 2);  
System.out.println(1 / 3);  
System.out.println(2 / 3);  
System.out.println(9 / 10);
```

these are all int/double, double/int, or double/double: as long as AT LEAST ONE of the values is a double, THEN the division results in a proper decimal value as expected!

```
System.out.println(10.0 / 5);  
System.out.println(10.0 / 4.0);  
System.out.println(10 / 3.0);  
System.out.println(3.0 / 2.0);  
System.out.println(1.0 / 3);  
System.out.println(2 / 3.0);  
System.out.println(9.0 / 10.0);
```

ACTUAL OUTPUT:

```
2  
2 (2.25 TRUNCATED to 2)  
3 (3.333.. TRUNCATED to 3)  
1 (1.5 TRUNCATED to 1 not rounded to 2)  
0 (0.333.. TRUNCATED to 0)  
0 (0.666.. TRUNCATED to 0 not rounded to 1)  
0 (0.9 TRUNCATED to 0 not rounded to 1)
```

```
2.0  
2.5  
3.3333333333333335  
1.5  
0.3333333333333333  
0.6666666666666666  
0.9
```

Solution ([back](#))

The issue is that both `totalPoints` and `earnedPoints` are **ints**, so dividing them results in **int division**: $55/60$ is 0.91666 but because they are **ints**, this decimal **GETS TRUNCATED TO 0!** 0 then gets multiplied by 100 to result in 0 (an **int**), and then stored in the **double** `testPercent` as 0.0, and printed.

To fix this, you could make `earnedPoints` or `totalPoints` or both **doubles** so that the division occurs between at least one double to ensure no truncation happens:

```
double earnedPoints = 55;
int totalPoints = 60;
double testPercent = earnedPoints / totalPoints * 100;
System.out.println("Test average: " + testPercent);
```

*recall that you **can** store an int into a double!*

Or

```
int earnedPoints = 55;
double totalPoints = 60;
double testPercent = earnedPoints / totalPoints * 100;
System.out.println("Test average: " + testPercent);
```

Or

```
double earnedPoints = 55;
double totalPoints = 60;
double testPercent = earnedPoints / totalPoints * 100;
System.out.println("Test average: " + testPercent);
```

Answer ([back](#))

Tyler *initially* tried to make the 100 a double, like shown below, but discovered that it didn't work (try it if you don't believe it!):

```
int earnedPoints = 55;
int totalPoints = 60;
double testPercent = earnedPoints / totalPoints * 100.0;
System.out.println("Test average: " + testPercent);
```

What was the error in Tyler's initial attempt to fix the program himself? (in other words, why doesn't the code above work as expected)?

because earnedPoints and totalPoints are still both ints, so the division between them (which occurs *first* left to right) is *still* int division, resulting in a truncated 0

Answer ([back](#))

Move the 100.0 (a double) to the *left* of the division, like this:

```
int earnedPoints = 55;
int totalPoints = 60;
double testPercent = 100.0 * earnedPoints / totalPoints;
System.out.println("Test average: " + testPercent);
```

Order of operations results in multiplication first, so `100.0 * earnedPoints` (double * int) goes first, resulting in a double value! *Then* division by `totalPoints` occurs, which is now a double / int operation, resulting in the correct output. **Sneaky!**