Using Operators and Math APIs

Arithmetic Operators and Promotion

Richard Monson-Haefel
Sr. Software Engineer

www.monsonhaefel.com

Overview

- Type of Operations
  - Arithmetic (+, -, *, /)
  - Assignment (=, +=, *=)
  - Comparison (>, <, ==)
  - Logical (&, &&, |, ||)
  - Byte Manipulation
    - Not covered
- Order of Operations
- Math APIs
  - random, round, pow, max, min

Order of Operations

Operator	Symbols
Post-Unary	*expr*++ *expr*--
Pre-Unary	++*expr* --*expr*
Other Unary	+*expr* -*expr* !*expr*
Multiplicative	* / %
Additive	+ -
Relational	< > <= >= instanceof
Equality	== !=
Logical	& ^ |
Logical (short-circuit)	&& ||
Ternary	expr ? expr : expr
Assignment	= += -+ *= /= %=

Order of Operations

Operator	Symbols
Post-Unary	*expr*++ *expr*--
Pre-Unary	++*expr* --*expr*
Other Unary	+*expr* -*expr* !*expr*
Multiplicative	* / %
Additive	+ -
Relational	< > <= >= instanceof
Equality	== !=
Logical	& ^ |
Logical (short-circuit)	&& ||
Ternary	expr ? expr : expr
Assignment	= += -+ *= /= %=

Pre- and Post-Unary Operators

Assignment Operators

Simple Assignment Operator

Assignment Operator as an Operation

```
x = 5;
y = 3; 8;

z = 5 + (y = x + y);
    5 + (y = 5 + 3)
    5 +    (y = 8)
    5 +        8

z = 13
```

Simple Assignment Operator

Assignment Operator as an Operation

```
boolean flag = false; true;
z = 0; 5;
if(flag == true){
    z = 5;
}else{
    z = 3;
}
```

Compound Assignment Operators

+=

-=

*=

/=

%=

Comparison Operators

Comparison Operators

==

!=

<

<=

>

>=

instanceof

Comparison Operators

==

!=

<

<=

>

>=

instanceof

Comparison Operators

==

!=

<

<=

>

>=

Numbers Only

instanceof

Comparison Operators

==

!=

Primitives numbers, Object References, and booleans

<

<=

>

>=

instanceof

Comparison Operators

==

!=

<

<=

>

>=

instanceof } Object Instances

Logical Operators

Logical Operators

&

&&

|

||

^

!

Logical Operators

&

&&

|

||

^

!

&

Both sides much be true

```
tru_1 = true;
tru_2 = true;
flse_1 = false;
```

& → Both sides must be true
    Both sides are tested

Logical Operators

&

&&

|

||

^

!

&&

Both sides must be true. Left side short-circuit if false

```
tru_1 = true;
tru_2 = true;
flse_1 = false;
```

```
& → Both sides must be true
    Both sides are tested
&&→ Both sides must be true
    IF left side = false
    THEN false
```

Logical Operators

&

&&

|

||

^

!

|

At least one side must be true

```
tru_1 = true;
true_2 = true;
flse_1 = false; flse_2 = false;
```

```
& → Both sides must be true
    Both sides are tested
&&→ Both sides must be true
    IF left side = false
    THEN false
|  → At least one side must
    be true
```

Logical Operators

&

&&

|

||

^

!

||

At least one side must be true. Left side short-circuit if true

```
tru_1 = true;
tru_2 = true;
flse_1 = false; flse_2 = false;
```

```
& → Both sides must be true
    Both sides are tested
&&→ Both sides must be true
    IF left side = false
    THEN false
| → At least one side must
    be true
||→ At least one side must
    be true
    IF left side = true
    THEN true
```

Logical Operators

&

&&

|

||

^

!

^

One side must be false and the other must be true

```
tru_1 = true;
true_2 = true;
flse_1 = false; flse_2 = false;
```

```
& → Both sides must be true
    Both sides are tested
&&→ Both sides must be true
    IF left side = false
    THEN false
| → At least one side must
    be true
||→ At least one side must
    be true
    IF left side = true
    THEN true
^ → One side must be false
    and the other side true
```

Logical Operators

&

&&

|

||

^

!

Logical Operators

&

&&

|

||

^

!

!

Reverses the boolean value; true → false , false → true

```
tru_1 = true;
true_2 = true;
flse_1 = false;


rst = !true_1;
rst = !(true_1 == false_1);
```

```
& → Both sides must be true
    Both sides are tested
&&→ Both sides must be true
    IF left side = false
    THEN false
| → At least one side must  be
true
||→ At least one side must  be
true
    IF left side = true
    THEN true
^ → One side must be false
    and the other side true
! → Reverses the boolean
```

The Ternary Operator

Ternary Operator

Similar to an if/else Statement

```
boolean result;
float x = (float)Math.random() * 6;

if(x <= 3){
    result = true;
}else{
    result = false;
}
```

```
> 2 --> true
> 4 --> false
> 1 --> true
```

Ternary Operator

Similar to an if/else Statement

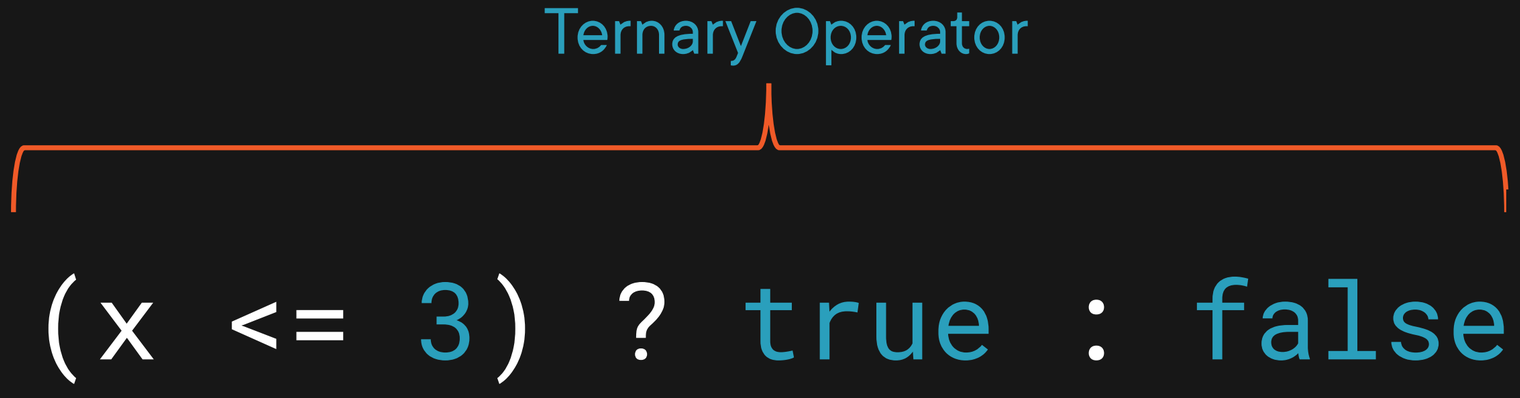
```
boolean result;
float x = (float)Math.random() * 6;
```

```
                    ?       :
```

```
> 2 --> true
> 4 --> false
> 1 --> true
> 4 --> false
> 0 --> true
> 6 --> false
```

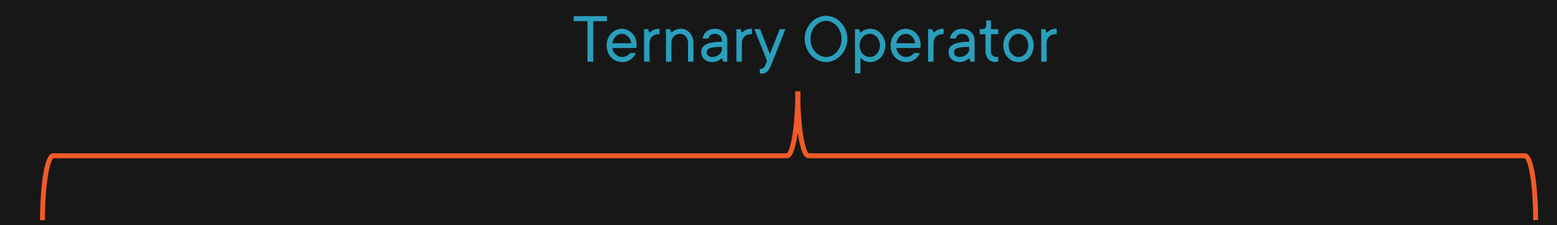
Ternary Operator

Similar to an if/else Statement



Ternary Operator

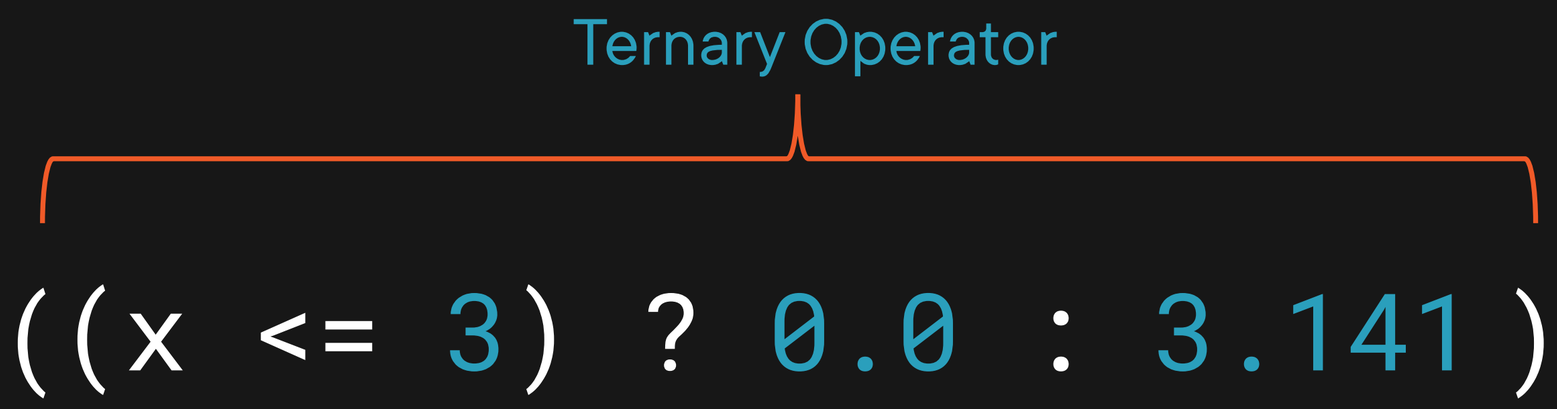
Similar to an if/else Statement




```
if( (x <= 3) ? true : false ){
    // do something
}
```

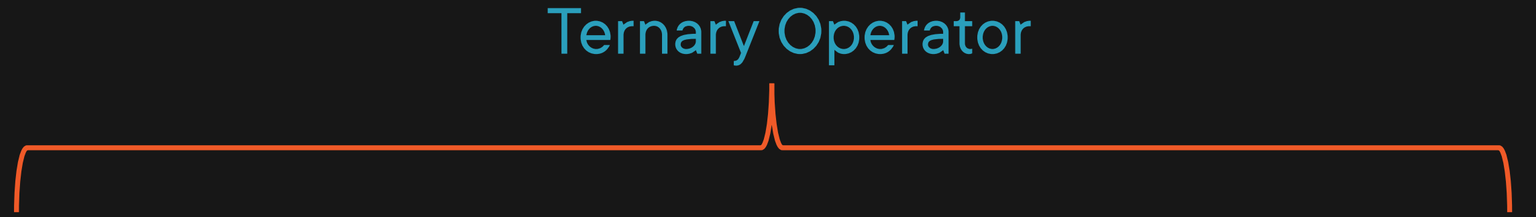
Ternary Operator

Similar to an if/else Statement



Ternary Operator

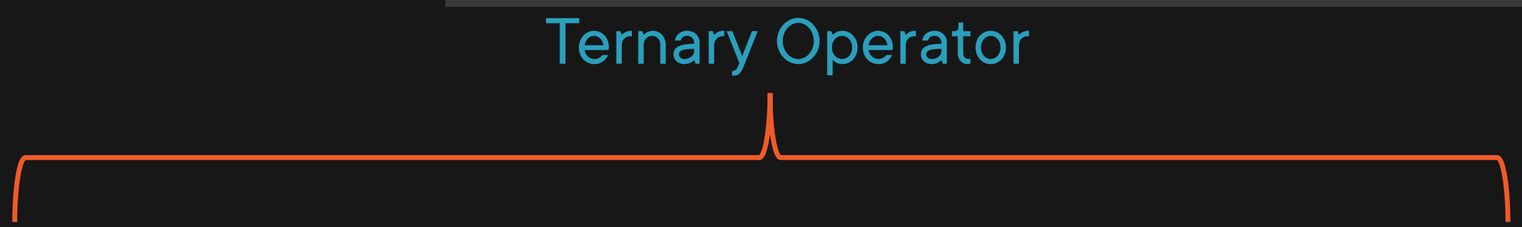
Similar to an if/else Statement




```
dValue = ((x <= 3) ? 0.0 : 3.141) * 13;
```

Ternary Operator

Similar to an if/else Statement




```
strValue = (x <= 3) ? x : “to high”
```

Ternary Operator

Similar to an if/else Statement

```
strValue = “The strValue is “ +
                (x <= 3) ? x : “to high.”
```


```
> The strValue is 2.5701542
> The strValue is to high.
> The strValue is 0.32083392
> The strValue is to high.
```

Other Operators

Bitwize Operators

&=

^=

|=

<<=

>>=

>>>=

Order of Operations: Part 1

Order of Operations

Operator	Symbols
Post-Unary	*expr*++ *expr*--
Pre-Unary	++*expr* --*expr*
Other Unary	+*expr* -*expr* !*expr*
Multiplicative	* / %
Additive	+ -
Relational	< > <= >= instanceof
Equality	== !=
Logical	& ^ |
Logical (short-circuit)	&& ||
Ternary	expr ? expr : expr
Assignment	= += -+ *= /= %=

Order of Operations

Unary Operators

m3.v8_OrderOfOperations.after

```
int x = 3;
int y = 4;

int z = x++ + y + --y + x;
        3     4     3     4
```

Symbols	
expr++	*expr*--
++*expr*	--*expr*

Order of Operations

Unary Operators

m3.v8_OrderOfOperations.after

```
int x = 3;
int y = 4;

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

3 + 3 + 3 + 4 = 14

Symbols	
expr++	*expr*--
++*expr*	--*expr*

Order of Operations

Unary Operator

m3.v8_OrderOfOperations.after

```
int x = 3;
int y = 4;

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

2 -2 +4

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*

Order of Operations

Unary Operator

m3.v8_OrderOfOperations.after

```
int x = 3;
int y = 4;

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

```
        2 + -2 +  4 = 4
```

Symbols
expr`++` `expr--`
`++`*expr* `--`*expr*
`+`*expr* `-`*expr* `!`*expr*

Order of Operations

Multiplicative and Additive

+

m3.v8_OrderOfOperations.after

```
int x = 3;
int y = 4;

int z = x * x++ + y - y / x
        3 * 3       4 / 4
```

Symbols
expr`++` *expr*`--`
`++`*expr* `--`*expr*
`+`*expr* `-`*expr* `!`*expr*
`*` `/` `%`
`+` `-`

Order of Operations

Multiplicative and Additive

m3.v8_OrderOfOperations.after

```
int x = 3;
int y = 4;

int z = x * x++ + y - y / x
```

3 * 3 4 / 4

9 4 1

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -

Order of Operations

Multiplicative and Additive

m3.v8_OrderOfOperations.after

```
int x = 3;
int y = 4;

int z = x * x++ + y - y / x
        3 * 3       4 / 4

        9   +   4   -   1   =   12
```

Symbols
expr`++` *expr*`--`
`++`*expr* `--`*expr*
`+`*expr* `-`*expr* `!`*expr*
`*` `/` `%`
`+` `-`

Order of Operations: Part 2

Order of Operations

Relational and Equality

m3.v9_OrderOfOperations.after

```
int x = 3, y = 4;
boolean zBool;

zBool = y + x * x > y & y != ++x;
            3 * 3           4
```

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=

Order of Operations

Relational and Equality

m3.v9_OrderOfOperations.after

```
int x = 3, y = 4;
boolean zBool;

zBool = y + x * x > y & y != ++x;
            3 * 3           4

        4   +  9
```

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=

Order of Operations

Relational and Equality

m3.v9_OrderOfOperations.after

```
int x = 3, y = 4;
boolean zBool;

zBool = y + x * x > y & y != ++x;
```

```
        3 * 3          4
    4 + 9
   13    >    4     4  !=  4
```

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=

Order of Operations

Relational and Equality

m3.v9_OrderOfOperations.after

```
int x = 3, y = 4;
boolean zBool;

zBool = y + x * x > y & y != ++x;
            3 * 3                4
        4 + 9
          13 > 4          4 != 4

               true  &  false = false
```

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=

Order of Operations

Logical

m3.v9_OrderOfOperations.after

```
boolean x = true;
boolean y = false

boolean z = x && y ^ x || y | x;
```

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
& ^ |
&&. ||

Order of Operations

Logical

m3.v9_OrderOfOperations.after

```
boolean x = true;
boolean y = false

boolean z = x && y ^ x || y | x;
```

true true

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
& ^ |
&&. ||

Order of Operations

Logical

m3.v9_OrderOfOperations.after

```
boolean x = true;
boolean y = false

boolean z = x && y ^ x || y | x;

          true && true ||  true
```

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
& ^ |
&&. ||

Order of Operations

Logical

m3.v9_OrderOfOperations.after

```
boolean x = true;
boolean y = false

boolean z = x && y ^ x || y | x;

          true && true || true

               true  ||  true = true
```

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
& ^ |
&&. ||

Order of Operations

Ternary and Compound Assignment

m3.v9_OrderOfOperations.after

```
x = 3;
y = 6;
z = 2;

z *= y/x - y - x > y ? 4 : 2;

          2 - 6
```

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
& ^ |
&& ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operations

Ternary and Compound Assignment

m3.v9_OrderOfOperations.after

```
x = 3;
y = 6;
z = 2;

z *= y/x - y - x > y ? 4 : 2;
```

2 - 6

-4

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
& ^ |
&& ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operations

Ternary and Compound Assignment

m3.v9_OrderOfOperations.after

```
x = 3;
y = 6;
z = 2;

z *= y/x - y - x > y ? 4 : 2;
          2 - 6
             -4 - 3
```

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
& ^ |
&& ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operations

Ternary and Compound Assignment

m3.v9_OrderOfOperations.after

```
x = 3;
y = 6;
z = 2;

z *= y/x - y - x > y ? 4 : 2;
```

2 - 6

-4 - 3

-7

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
& ^ |
&& ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operations

Ternary and Compound Assignment

m3.v9_OrderOfOperations.after

```
x = 3;
y = 6;
z = 2;

z *= y/x - y - x > y ? 4 : 2;
        2 - 6
          -4 - 3
            -7 > y ? 4 : 2
```

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
& ^ |
&& ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operations

Ternary and Compound Assignment

m3.v9_OrderOfOperations.after

```
x = 3;
y = 6;
z = 2;

z *= y/x - y - x > y ? 4 : 2;
```

```
2 - 6
-4 - 3
-7 > y ? 4 : 2
2
```

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
& ^ |
&&. ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operations

Ternary and Compound Assignment

m3.v9_OrderOfOperations.after

```
x = 3;
y = 6;
z = 2;

z *= y/x - y - x > y ? 4 : 2;
```

```
2 - 6
-4 - 3
-7 > y ? 4 : 2
z *= 2
```

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
& ^ |
&&. ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operations

Ternary and Compound Assignment

m3.v9_OrderOfOperations.after

```
x = 3;
y = 6;
z = 2;

z *= y/x - y - x > y ? 4 : 2;
```

```
2 - 6
-4 - 3
-7 > y ? 4 : 2
z = 2 * 2
```

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
& ^ |
&& ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operations

Ternary and Compound Assignment

m3.v9_OrderOfOperations.after

```
x = 3;
y = 6;
z = 2;

z *= y/x - y - x > y ? 4 : 2;
```

```
2 - 6
-4 - 3
-7 > y ? 4 : 2
```

```
z = 4
```

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
& ^ |
&& ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operations

Ternary and Compound Assignment

m3.v9_OrderOfOperations.after

```
x = 3;
y = 6;
z = 2;

z *= y/x - y - x > y ? 4 : 2;
     2 - 6
       -4 - 3
        -7 > y ? 4 : 2
z = 4
```

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
& ^ |
&& ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operation: Part 3

Order of Operations

Parentheses

m3.v9_OrderOfOperations.after

```
int x = 3, y = 4;

int z = --x * x + y + 8;
          2 * 2
            4   + 4 + 8 = 16
```

Symbols
expr++ *expr*--
++*expr* --*expr*
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
&&. ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operations

Parentheses

m3.v9_OrderOfOperations.after

```
int x = 3, y = 4;

int z = --x * x + y + 8;
          2 * 2
            4   + 4 + 8 = 16
```

Symbols
expr++ *expr*--
++*expr* --*expr*
( )
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
&&. ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operations

Parentheses

m3.v9_OrderOfOperations.after

```
int x = 3, y = 4;

int z = --x * x + y + 8;
          2 * 2
            4   + 4 + 8 = 16
```

```
x = 3; y = 4;

z = --x * (x++yy++88);
```

Symbols
expr++ *expr*--
++*expr* --*expr*
( )
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
&&. ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operations

Parentheses

m3.v9_OrderOfOperations.after

```
int x = 3, y = 4;

int z = --x * x + y + 8;
          2 * 2
            4   + 4 + 8 = 16
```

```
x = 3; y = 4;

z = --x * (x + y + 8);
        2   *   14 = 28
```

Symbols
expr++ *expr*--
++*expr* --*expr*
( )
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
&&. ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operations

Parentheses

m3.v9_OrderOfOperations.after

```
int x = 3, y = 4; z = 0;

int z = (--x * x + (y + x) - y--);

int z = --x * ((x + y) + x) - y--;

int z = (--x * x + y + (x - y--));
```

Symbols
expr++ *expr*--
++*expr* --*expr*
( )
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
&&. ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operations

Parentheses

m3.v9_OrderOfOperations.after

```
int x = 3, y = 4; z = 0;

int z = --x * x + (y + x) - y--);

int z = (--x * (x + (y + x - y--;

int z = (--x * x + (y + x)) - y--);
```

Symbols
expr++ *expr*--
++*expr* --*expr*
( )
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
&&. ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operations

Parentheses

m3.v9_OrderOfOperations.after

```
int x = 3, y = 4; z = 0;

int z ( = --x * x + (y + x) – y--);

int z = --x (* (x + y) + x – y--);

int z = (--x * x + (y + x)) – y--);)
```

Symbols
expr++ *expr*--
++*expr* --*expr*
( )
+*expr* -*expr* !*expr*
* / %
+ -
< > <= >= instanceof
== !=
&&. ||
expr ? *expr* : *expr*
= += -+ *= /= %=

Order of Operations

Operator	Symbols
Post-Unary	*expr*++ *expr*--
Pre-Unary	++*expr* --*expr*
Brackets	( )
Other Unary	+*expr* -*expr* !*expr*
Multiplicative	* / %
Additive	+ -
Relational	< > <= >= instanceof
Equality	== !=
Logical	& ^ |
Logical (short-circuit)	&& ||
Ternary	*expr* ? *expr* : *expr*
Assignment	= += -+ *= /= %=

PUMA is a REBL TA

PUMA is a REBL TA

P – Pre-/Post-Unary
U - Unary
M - Multiplicative
A – Additive

R- Relational
E- Equality
B- BitWise
L- Logical (logical && > logical ||)

T- Ternary
A- Assignment

Math APIs

Math APIs

random( )

round( )

pow( )

max( )

min( )

Summary

- Four Categories of Operations
  - Arithmetic (+, -, *, /)
  - Assignment (=, +=, *=)
  - Comparison (>, <, ==)
  - Logical (&, &&, |, ||)
- Order of Operations
  - Parentheses
  - PUMA is a REBL TA
- Math APIs
  - random, round, pow, max, min

Order of Operations

Operator	Symbols
Post-Unary	*expr*++ *expr*--
Pre-Unary	++*expr* --*expr*
Brackets	( )
Other Unary	+*expr* -*expr* !*expr*
Multiplicative	* / %
Additive	+ -
Relational	< > <= >= instanceof
Equality	== !=
Logical	& ^ |
Logical (short-circuit)	&& ||
Ternary	*expr* ? *expr* : *expr*
Assignment	= += -+ *= /= %=

PUMA is a REBL TA

P – Pre-/Post-Unary
U - Unary
M - Multiplicative
A – Additive

R- Relational
E- Equality
B- BitWise
L- Logical (logical && > logical ||)

T- Ternary
A- Assignment

Up Next:
Using Primitive Wrappers