Excitement Postponed

- Sorry. Again, this stuff is vital.

- Promise we’ll get to cool stuff.

  - But actually, this stuff is pretty neat.
  - And powerful…

  - And quite possibly the basis of life as we know it.
    - Ok. I’m exaggerating now.
Arithmetic Operations

- Can do math in a program
  - Basic operators
    - +, -, /, *
    - modulo %
    - increment ++, decrement --
    - assignment operators +=, -=, *=, /=
  - Also have booleans … !, &&, ||
  - And host of less common.
    - Will see all these as they come up
Basic Math

- Use just like in a math class
  - Use * for multiply

```java
double a = 2.0, b = 2.4, c = 16.0;
double doubleNum = c/a - b + 2.3 ;  //gives 7.9

int i = -1, j = -2, intNum;
intNum = i * j;                    //gives 2

double answer = doubleNum - intNum; //gives 5.9
```

Implicit casting (bad)!
Casting Revisited

- Double *always* wins
  - If a double is in the equation it converts anything else to a double

- But why be confusing?
  - These are the same
    - `double answer = doubleNum – intNum;`
    - `double answer = doubleNum – (double) intNum;`
      - This is much clearer!
Integer Division

What if you divide two integers?

- **Throws away the remainder.**
  - 9/4 is 2
  - 21/14 is 1
  - 9/3 is 3
  - 500/99 is 5

- What if I want the remainder?
  - modulo!
Modulo

- Modulo is the remainder after division.
  - (Kind of the opposite of integer division.)
  - Only works on integers!
    - 5%3 is 2
      - Why? 5 divided by 3 has a remainder of 2
    - 20%10 is 0
    - 13%4 is 1
    - 11 % -3 is 2
      - Why? -3 divides into 11 a total of -3 times with a remainder of 2
  - 7 % 0 is infinite! (remainder after division by 0)
  - 3.2 % 7 is meaningless!
int numberOfDonuts = 63;

//find how many full boxes of a dozen each
int numberOfDozens = numberOfDonuts / 12;

//how many left over (after boxed into dozens)
int leftover = numberOfDonuts % 12;

//how many half dozen boxes
int numberOfHalfDozens = leftover / 6;

//how many left over (after half dozen boxes)
leftover = leftover % 6;

why no int?

integer division!

Work through the numbers. You’ll see.
Increment And Decrement

- Very frequently want to count up and down.
  - Java, C++, and others make easy.
    - i++ increases i by 1
    - i-- decreases i by 1

```java
int countdown = 10;
while(countdown > 0)
{
    countdown--;
    System.out.println(countdown);
}
```

Can you think of a nicer way to display?
```java
(JOptionPane.showMessageDialog(null, countdown));
```
Assignment Operators (Why Need)

- Have you noticed we keep writing
  - countdown = countdown - 1;
  - or age = age + 1;
  - or…

  - Nicer choice
    - countdown--
    - age++

- But what if count by two’s?
  - numBicycleTires = numBicycleTires + 2;
Assignment Operators (How Use)

Try this!
- numBicycleTires += 2;
  - same as numBicycleTires = numBicycleTires + 2;

Can use +=, -=, *=, /=, %=

```java
int i = 3;
i += 7;
i -= 4;
i /= 5;
i *= 100;
i %= 91;
```

What’s the final value of i?
Answer: 9
Order of Operations (part 1)

Which operation goes first?

- Does it matter?

```java
double a = 2.0, b = 3.0, c = 4.0, d;
    d = a * b - 2.0 / c;
```

- If it is left to right, this is 1.0
- if multiplication, division, subtraction (in that order) this is 5.5

- Wow, radical difference!
Order of Operations (part 2)

Rules

1. Parentheses come first ()
2. Then unary operators ++, --
3. Then *, /, %
4. Then +, -
5. If two have same precedence, then left to right
Order of Operations (part 3)

Find the value now…

double a = 2.0, b = 3.0, c = 4.0, d;
d = a * (b - 2.0) / c;

1. parentheses first
   \[ d = a * 1.0 / c \]
2. *, /, % next (left to right)
   \[ d = 2.0 / c \]
   \[ d = 0.5 \]

How about…

d = ((a * b) - 2.0) / c;
Order of Operations (part 4)

- Use parentheses liberally!
  - They make equations clearer.
  - Analog of {...} when using loops.

- Don’t have to worry about order of operations if you use parentheses!
  - \( d = ((a \times b) - 2.0) / c; \)
  - \( d = (a \times b) - (2.0 / c); \)
  - \( d = a \times ((b - 2.0) / c); \)

- Each of these has only one possible interpretation!