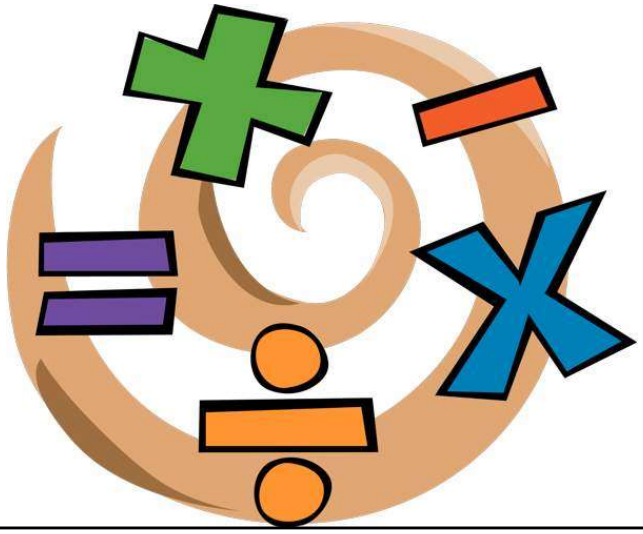
Programming Rules, Errors, and Debugging



`final` variables

If a variable is declared `final`, then after its value is first set, it cannot be changed.

This lets us define useful constants once and then use them repeatedly, without worrying that someone might change them

```
final double PI = 3.14159265;
```

Someone like 'ourselves' by accident

Syntax vs. Semantics

The *syntax rules* of a language define how we can put together symbols, reserved words, and identifiers to make a valid program

The *semantics* of a program statement define what that statement means (its purpose or role in a program)

A program that is syntactically correct is not necessarily logically (semantically) correct

A program will always do what we tell it to do, not what we hoped to tell it to do

Errors

A program can have three types of errors

The compiler will find syntax errors and other basic problems (*compile-time errors*)

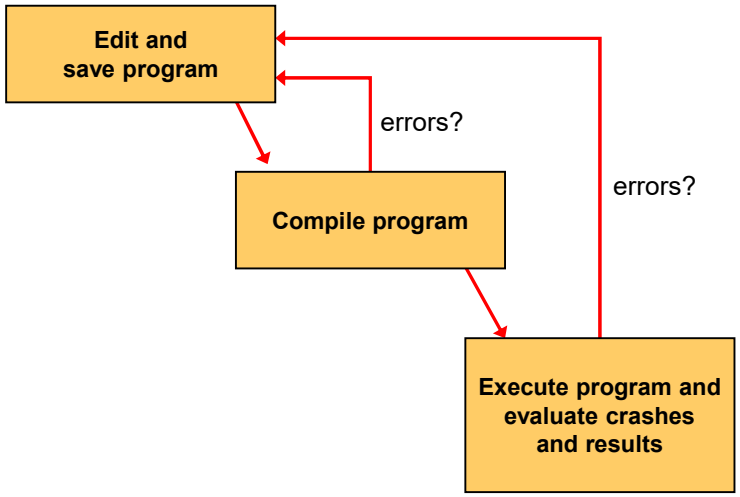
- If compile-time errors exist, an executable version of the program is not created

A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (*run-time errors*)

A program may run, but produce incorrect results, perhaps using an incorrect formula (*logical errors*)

"crash" = terminate abnormally

Basic Program Development



Awfully nice when compiler finds errors for us.

Java Math operations

The basics…

- +, -, *, /, % are built in

Some weirdness

- byte + byte = int
- short + short = int
- int + int = int
- But long + long = long

% is remainder. Only works with integer data types.

Java Math operations

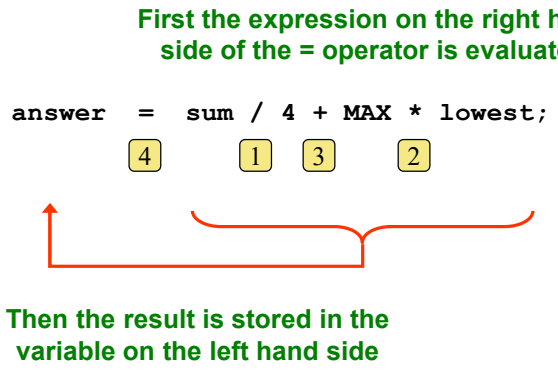
- `short price = 10;`
- `short tip = 2;`
- `short total = (short) price + tip; // ERROR:  why?`

Order of Operations! Remember PEMDAS?

- Parentheses, Exponentiation, Multiplication and Division, Addition and Subtraction
- Casting happens before addition and multiplication
- `short price = 10;`
- `short tip = 2;`
- `short total = (short) (price + tip); // Works!`

Assignment

The assignment operator "=" has a lower precedence than the arithmetic operators



Assignment

The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the original value of `count`

```
count  =  count + 1;
```

Then the result is stored back into `count` (overwriting the original value)

Java Math Operations

Can combine a built-in operation and assignment with "hybrid" operators

- `+=, -=, *=, /=, %=`
- `int counter = 0;`
- `counter += 2;`

- `int x=11, y=5;`
- `x %= y`

```
// What is value of x?
```

Java Math Operations

Increment and decrement operators: ++, --

If used before variable ("prefix operator"), then variable value is changed before used. If used after variable ("postfix operator"), value is used before changed.

```
long userId = 100L;
long myId = ++userId;
System.out.println("myId: " + myId + ", userId: " + userId);
// What do we get?

long newId = userId++; // equals 101, but userid is now 102
System.out.println("newId: " + newId + ", userId: " + userId);
// What do we get?
```

Java order of operations:
- Prefix/Parentheses/Casting/Multiply,Divide/Add,Subtract/Assignment/Postfix

Details on numerical constants

Type of integer constants such as -17 or 259 is int

If we want a long constant, we put an 'L' afterwards, e.g.

```
long bigNumber = 9876543210L;
```

Type of floating point constants such as 1.4141 or -0.25 is double

If we want a float constant, we put an 'F afterwards, e.g.

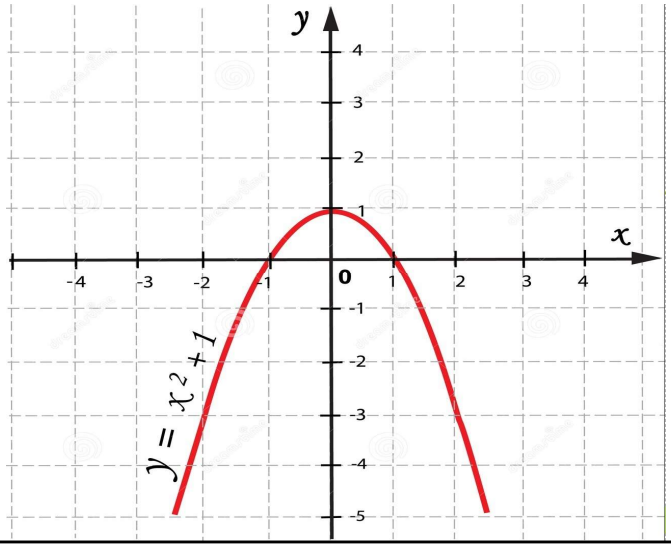
```
float oneFifth = 0.2F;
```

We can also use a cast. e.g.

```
float oneFifth = (float) 0.2; long bigNumber = (long) 9876;
```

CS112 –
Java
Programming

Java Math library



Java Math functions

The **Java Math library** includes many useful functions for working with numbers.

Online-search "java math" for reference or for specific functions

Some useful functions:

- Trig: sin(), cos(), tan(), asin(), acos(), atan(), etc
- Powers: sqrt(), pow(), exp(), log(), log10()
- Conversions: abs(), ceil(), floor(), max(), min(), round()
- Constants: Math.E, Math.PI

Java Math Functions

```
double angle = Math.PI / 2.0;

System.out.println(Math.sin(angle)); // angle in RADIANS
```

Math Conversion Functions

```
Math.ceil(2.1); // "next biggest" integer
```

3.0

```
Math.ceil(3.0);
```

3.0

```
Math.round(2.5);
```

3.0

```
Math.round(-2.5);
```

-2.0

CEILING(): "smallest integer that is >= argument"
FLOOR is similar

Round() rounds "towards +infinity"

Suppose we want to round but not to 1?

```
// Want PI to 3 decimal places

final double scalefactor = 1000.0;

double roundedPi = Math.round(Math.PI * scalefactor) / scalefactor;

System.out.println(roundedPi);
```

3.142

Boolean variables and Expressions

Boolean Expressions

Comparisons give Boolean results:

- >, >=, <, <=, ==, !=

```
boolean flag1 = 1 < 2;
boolean flag2 = (3.14159 > Math.PI);
```

As in Python, '=' is assignment, '==' is comparison.

```
flag5 = flag3 == flag4;
```

Review comparison operators: ==, !=

Boolean Expressions

Logical values can be "operated on": &&, ||

```
boolean flag3 = (flag1 || flag2);
```

Logical 'NOT': !

```
boolean flag4 = !flag3;
```

Hybrid operators "operate-and-assign": &=, |=

```
flag5 &= flag2; // flag5 = flag5 && flag2
```

Review logical operations: and, or
SLOW WALKTHRU ON HYBRID EXAMPLE!

booleans often used to control loops

Much more on loops later, but basic while() loop:

```
int counter = 0;
while (counter < 10) { // curly brace starts block of code
    System.out.println("Counter is " + counter);
    counter++;
} // end of while() loop
```

FORMALLY, while() executes the following statement or block
ALWAYS use curly braces to define a code block. NEVER just 1 statement
BE CAREFUL to make sure your loop will terminate eventually

What is the order of operations for Boolean operators?

All logical operators have lower precedence than the comparison operators

The ! operator has higher precedence than && and ||

- Comparison: >, < , >=, <=, ==, !=
- Logical: !
- Logical: &&, ||

CS112 –
Java
Programming

Spring 2024

Loops and Conditionals

if()-else statement

```
class ProgramInput {
     static public void main(String[] args) {
          int x = 3;
          int y = 500;
          if (x > y) {
               System.out.println("x is larger");
          } else {
               System.out.println("y is larger");
          }
     }
}
```

Controlled via boolean expression
"else" clause is optional

Brief…

Good style: ALWAYS USE CURLY BRACES

The Conditional Operator

The conditional operator is similar to an `if-else` statement, except that it is an expression that returns a value

For example:

```
larger = ((num1 > num2) ? num1 : num2);
```

If `num1` is greater than `num2`, then `num1` is assigned to `larger`; otherwise, `num2` is assigned to `larger`

The conditional operator is *ternary* because it requires three operands

The Conditional Operator

Another example:

```
System.out.println("Your change is " + count +
   ((count == 1) ? "Dime" : "Dimes"));
```

If `count` equals 1, the "Dime" is printed

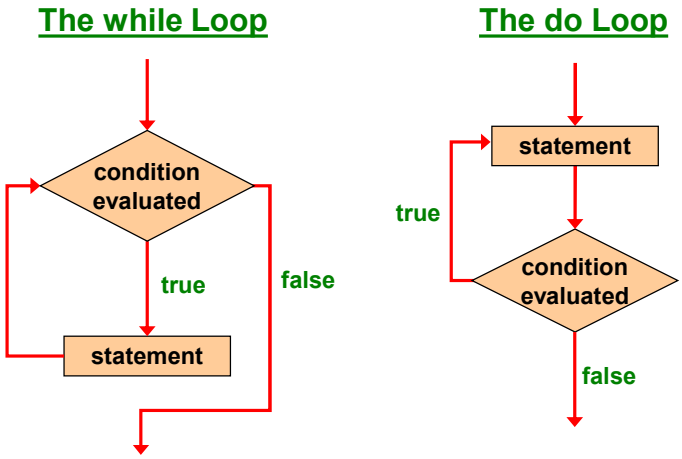
If `count` is anything other than 1, then "Dimes" is printed

The switch Statement

```
switch (option)
{
   case 'A':
      aCount++;
      break;
   case 'B':
      bCount++;
      break;
   default:
      cCount++;
      break;
}
```


What types can we "switch on"?
What does "break" do?
What if no "default"?

Comparing `while` and `do` loops



The for Statement

An example of a `for` loop:

```
for (int count=1; count <= 5; count++) {
   System.out.println(count);

}
```

The initialization section can be used to declare a variable

Like a `while` loop, the condition of a `for` loop is tested prior to executing the loop body

Therefore, the body of a `for` loop will execute zero or more times

Useful to run a fixed number of iterations, with an index variable

The for Statement

Each expression in the header of a `for` loop is optional

If the initialization is left out, no initialization is performed

If the condition is left out, it is always considered to be true, and therefore creates an infinite loop

If the increment is left out, no increment operation is performed

Nested Loops

```
for (int i= 0; i < height; i++) {
 for(int j = 0; j < width; j++) {
      totalBrightness +=image.pixelValue(i, j);
 }
}
```

Start with 0, end with one-less-than size. Because that is how array indexing works!

WHAT DOES break DO INSIDE A LOOP? EXITS FROM INNERMOST LOOP.

Boolean "early termination"

```
double numer = -1.0, denom = 0.0;

if (denom != 0.0 && (numer/denom) > 1.0) {
      System.out.println("Value is greater than 1");
}

if (numer < 0.0 || (numer/denom) < 0.0) {
      System.out.println("Error");
}
```

Since denom == 0.0, Java never evaluates (numer/denom) for the “&&” case.
Since numer < 0, Java never evaluates (numer/denom) for the "||" case.

How about this?

```
int y = 10, z = 5;
for (int x = 0; y > 0; z = x) {
      x++;
      y = z - x;
}
```

WALK THROUGH
AWFUL CODE: NEVER DO THIS. Try to have only a single variable inside for loop, and change and check only it

Summary

Conditionals

- if else-if else
- "conditional"
- switch-case-break

Loops

- while
- do-while
- for