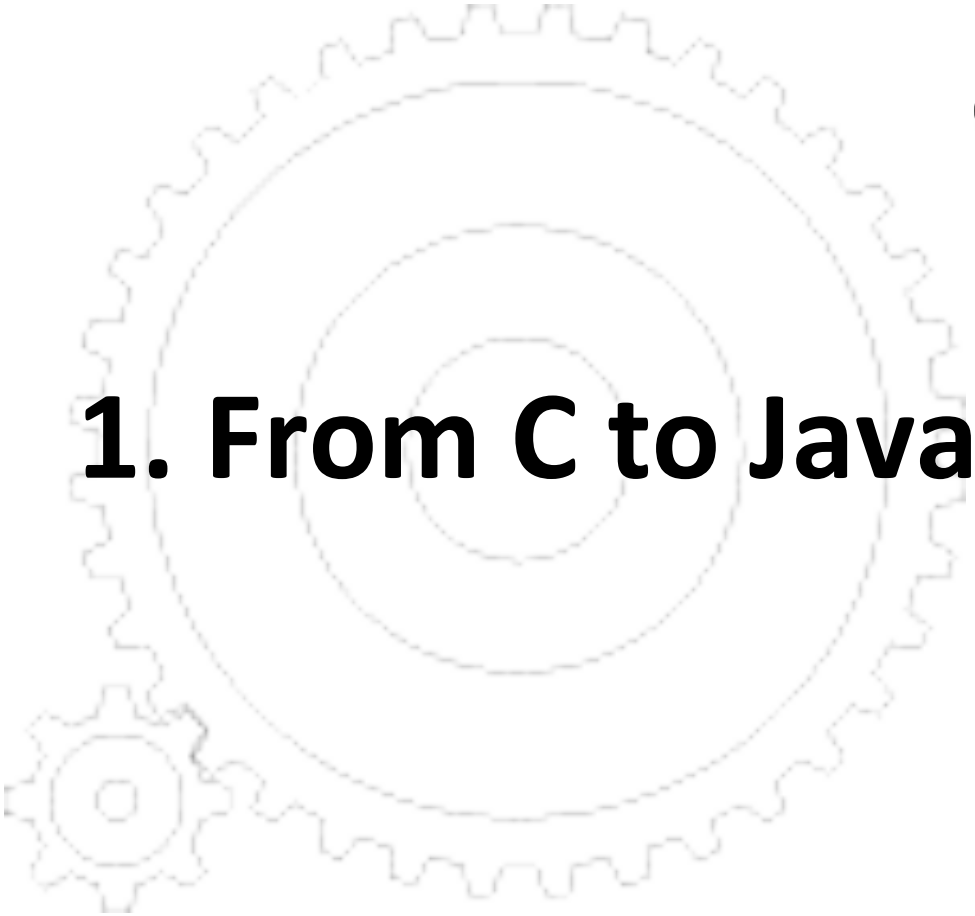
G51OOP

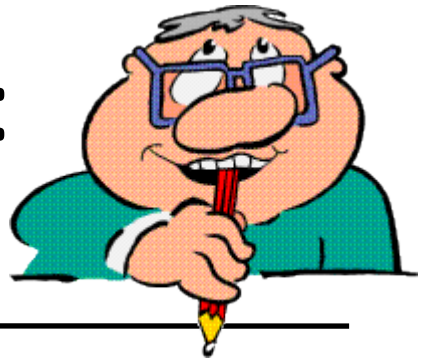


1. From C to Java

Object Oriented Programming

Colin Higgins

A simple program :



Filename: HelloWorld.java

```
// Program written by azt
/* September 1998 */

import java.lang.*;

/** Example class HelloWorld */
public class HelloWorld {
     public static void main(String argv[] ) {
          System.out.println("Hello World!");
     }
}
```

comments

method body

class body

Program Output

```
// to print more complex items
System.out.println("Hello " + "World!!!");

// or you could write
System.out.print("Hello ");
System.out.println("World!!!");

// you could write
System.out.println("Colin Higgins");
System.out.println("School of Computer Science and IT");
System.out.println("University of Nottingham");

// or you could write
System.out.println("Colin Higgins\nSchool of Computer Science
\nUniversity of Nottingham");
```

Constants and variables and Identifiers

Very similar to C

A Simple Example


```
import java.io.*;

class Add2Numbers {
    final float PI = 3.14F;

   public static void main(String argv[]) {
       double a, b, c;  // declare variables
       a = 1.75;     // assign values
       b = 3.46;
       c = a + b;        // add them together
       System.out.println("sum = " + c);
       System.out.println("Pi = " + PI );
   }
} // end class Add2Numbers
```

Primitive Types

- The basic Java types are given below along with their use, number of bytes required and their ranges :

1.	boolean	Logic	1 bit	true : false
2.	byte	8-bit signed integer	1 byte	-128 : +127
3.	short	16-bit signed integer	2 bytes	-32768 : +32767
4.	int	32-bit signed integer	4 bytes	-2147483648 : +2147483647
5.	long	64-bit signed integer	8 bytes	-2^{63} : $2^{63}-1$
6.	char	16-bit unsigned integer, representing Unicode chars	2 bytes	0 : 65535
7.	float	Single precision IEEE 754 floating point	4 bytes	1.4e-45 : 3.4e+38
8.	double	Double precision	8 bytes	4.9e-324 : 1.8e+308

Program layout

This is extremely important.

- Use ctrl-shift-f in Eclipse frequently!
- Remember humans are good at pattern recognition so use this to minimise errors!
- Layout your program carefully.
- Leave plenty of (but not too much) white space.
- Indent as appropriate.
- Use one of the recommended conventions. (eg brace position)
- Stick to the same convention.
- As programs become bigger, layout becomes more and more important.
- More later!

Operators

Arithmetic :

```
()
* / %
+ -
```

Comparison:

```
>   // greater than
<   // less than
>=  // greater than or equal to
<=  // less than or equal to
==  // equals - watch for the double "="!!!
!=  // not equals
```

More Operators

Logical:

```
&&  // conditional and
||  // conditional or
!   // logical complement (not)
&   // boolean and
|   // boolean or
^   // boolean exclusive or
```

Incremental:

```
++i // increment, deliver new value
i++ // increment, deliver old value
--i // decrement, deliver new value
i-- // decrement, deliver old value
```

Assignment

Assignment is also an operator.

There are many types of assignments – Java like C/ C++ is rich in operators.

```
int i; i = 3;
float f = 0.3456;
char ch = 'A';
```

assignment can be combined with other operators :

```
i += 4;   // plus-and-becomes ie i = i + 4
i -= j;   // minus-and-becomes ie i = i - j;
f *= 4.2; // multiply-and-becomes ie f = f * 4.2;
a &= b;   // logical &-and-becomes ie a = a & b;
i += 1;   // normally you would see i++;
```

Program input – simple way

There are many different ways of inputting strings, chars, ints, floats, etc

It is best for now to use our G51OOPInput class.

To read a long from the keyboard use…

```
long number;
number = G51OOPInput.readLong();
```

Or

```
G51OOPInput.promt("please type a long");
number = G51OOPInput.readLong();
```

Also available…

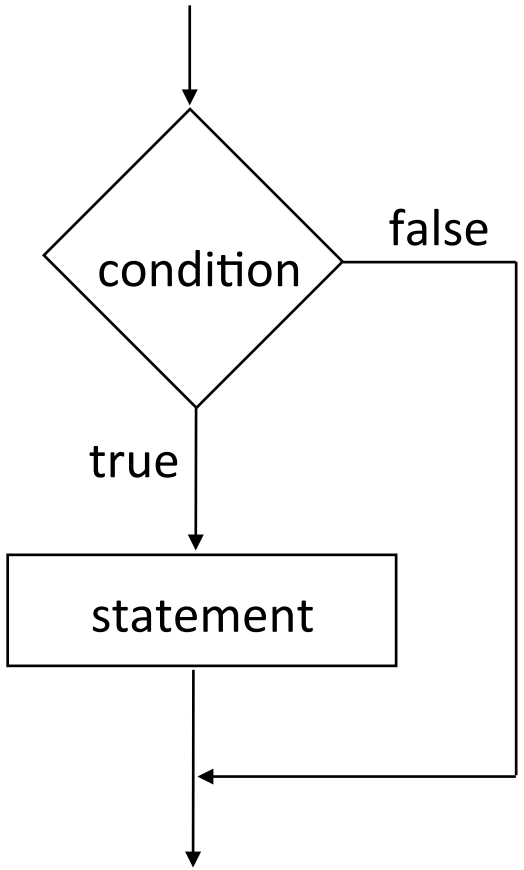
```
readChar()  readString()    readShort()
readFloat() readDouble()
```

Conditionals

- Simple "`if`" statement
- "`if`" … "`else`" statements
- Further Alternatives
- Possible Conditions

- Simple "`switch`" statements
- "`switch`" Example
- "`case`" values

Simple "*if*" statements



- The "*if*" statement, provides decision making capabilities.
- It causes selected statements to be executed if a certain condition evaluates to true at that point in the program.

Syntax:

```
if (condition)
        statement;
```

- If the boolean condition is true, the statement is executed; if it is false, the statement is skipped

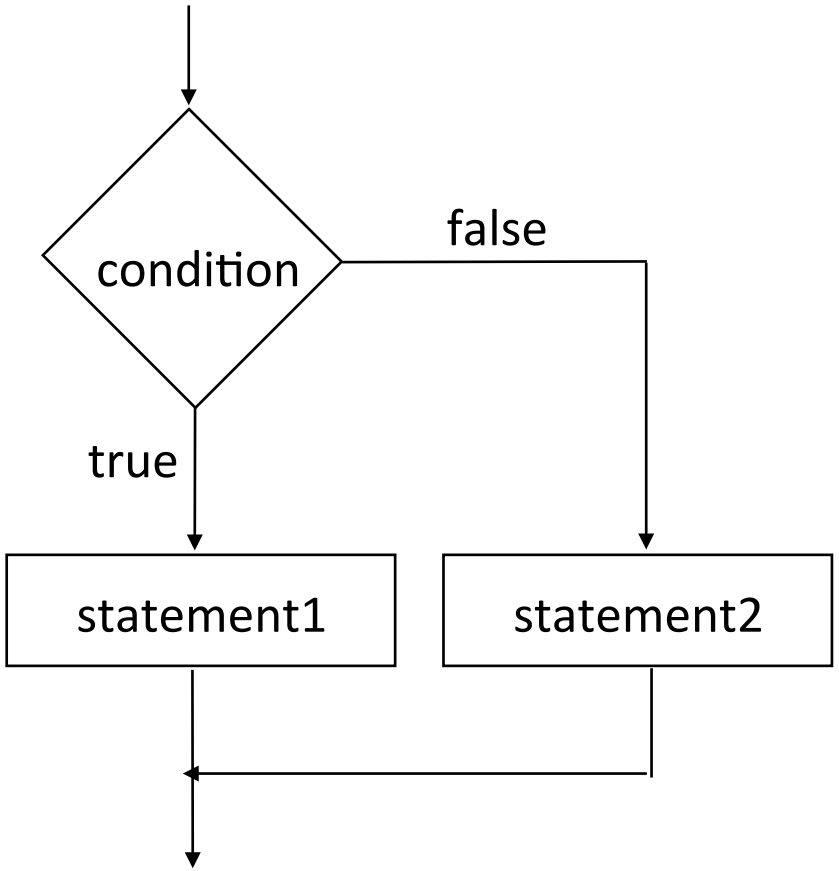
Simple “*if*” statements

```
int radius, result = 0;
radius = G51OOPInput.readInt(); // user code

if ( radius > 0 ) {
    result = radius * radius;
    System.out.println("radius is positive");
}   // end if radius > 0

System.out.println(radius + " " + result);
```

"*if ... else*" statements



- If the condition does not hold, we may wish to execute some different statements as alternatives to the "if" statements.

Syntax:

```
if (condition)
  statement;  else
  statement;
```

"*if ... else*" statements

```
int money;
int deposits = 0 , withdrawals = 0;
money = G51OOPInput.readInt();
if ( money > 0 ) {
    System.out.println("Deposit.");
    deposits = money + deposits;
} else {
    // if money <= 0
    System.out.println("withdrawal.");
    withdrawals = money - withdrawls;
}  // end if else money > 0
System.out.println("Transaction noted.");
```

Further alternatives

- We can add further alternatives to an "if" statement if we require.

```
if ( radius > 100 ) {        // radius > 100
    ....;
} else if ( radius > 10 ) { // radius > 10 and radius <= 100
    ....;
} else if ( radius > 1 ) {  // radius > 1 and radius <= 10
    ....;
} else {                    // radius <= 1
    ....;
}                           // end if radius various values
```

- The tests in the if statements are executed exactly in the order in which they are encountered. The first test here is “radius > 100"; if this is false, the second test if executed, testing whether “radius > 10", so that this is effectively the test “radius <= 100 && radius > 10", since we know that the first test is false.

"switch" statements

- `if` statements give a choice between two alternatives.
- May need a choice between more possibilities.
- The "`switch`" construct allows for any number of different actions to be taken dependent on the value of an integer calculation.
- The syntax of the `switch` statement is:

```
switch (expression) {
   case value1:
      statement
   case value2:
      statement
   case …
}
```

"`case`" Values

- The value after the word "`case`" must be a constant, you could not put case PartType, where PartType is a variable. You must put an explicit constant preferably via a declared final, for example :

```
static final int INCHES_PER_FOOT = 12;
```

- Two "case" labels can be adjacent.
- Don't forget the "`break;`" statements where you need them. You will normally want control to leave the "`switch`" statement at the end of each separate "`case`".
- The "`default:`" entry is optional, but should normally be included, even if only to report an error.
- The expression you switch on must be one of int, byte, char, short or long.

"switch" example

```
static final int STATE_A = 0;
static final int STATE_B = 42;
etc
int inputValue;

inputValue = G51OOPInput.readInt();
switch ( inputValue ) {
    case STATE_A:
    case STATE_B:
        myVariable = x * y;
        break;
    case STATE_C:
        myVariable = x * y + 4;
        break;
     case STATE_C:
        myVariable = x * y + 7;
        break;
    default:
         myVariable = 0;
}
```

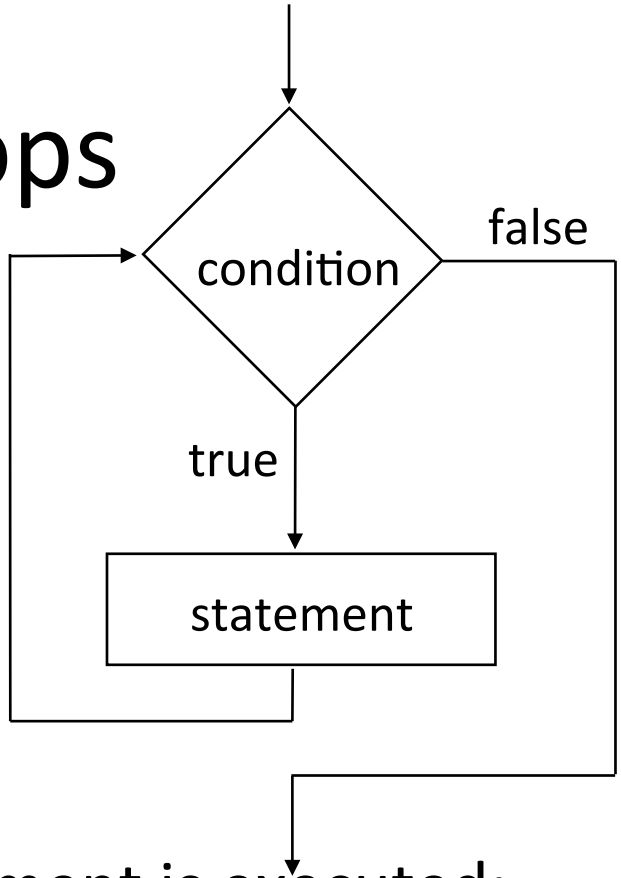
Character "switch" example

```
char commandChar;
commandChar = G51OOPInput.readChar();
switch ( commandChar ) {
case 'e':              // really should use static final…
    // somehow do an edit here
    break;
case 'l':
case 'p':
    // somehow do a print here
    break;
default:
    System.out.println("Don't understand " + commandChar);
} // end switch( commandChar )
```

Loops

- In the programs we have written so far, the statements in the program have been executed in sequence, from the start of the program to the end, omitting sections of "`if`" and "`switch`" constructs which have not been selected. The real power of computers comes from their ability to execute given sets of statements many times, commonly known as looping.
- In Java we have four types of loops :
  - The `while` loop
  - The `do while` loop
  - The `for` loop
  - (The `for` each loop)

"while" loops



- A *while statement* has the following syntax:

```
while (condition)
   statement;
```

- If the condition is true, the statement is executed; then the condition is evaluated again
- The statement is executed over and over until the condition becomes false

"while" loops

```
int number = 10;
while ( number >= 0 ) {
   System.out.println( "number is " + number );
   number--;
}
System.out.println( "Loop ended");
```

- The condition to be tested is contained in parentheses (round brackets) after the word "while", and the body of the loop is in curly braces after the condition.
- There is no semicolon after the closing curly brace.

Examples of while loops counting from 1 to 10

```
// version 1
int number = 1;
while ( number <= 10 ) {
   System.out.println( number );
   ....;
   number++;
}
```

```
//version 2
int number = 0;
while ( number < 10 ) {
   number++;
   System.out.println( number );
   ....;
}
```

```
// Version 3
int number = 0;
while ( number++ < 10 ) {
   System.out.println( number );
   ....;
}
```

```
// Version 4
int number = 0;
while ( ++number <= 10 ) {
   System.out.println( number );
   ....;
}
```

Examples of "while" loops

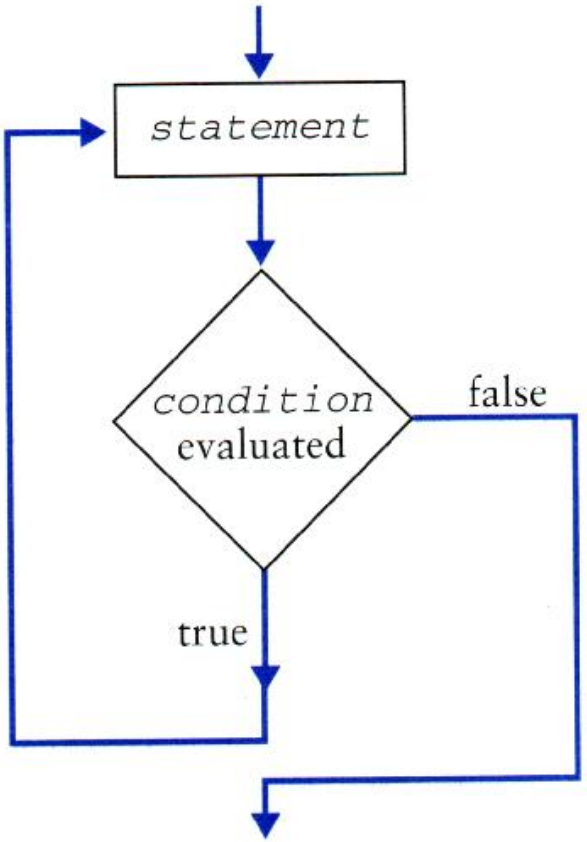
To add together the sequence 1 + 1/2 + 1/4 + 1/8 + ...
until the terms we are adding together are smaller than 0.00001

```
float term = 1.0f, total = 0.0f;
int counter = 0;
final float DELTA = 0.00001f;
while ( term > DELTA ) {
    total += term;
    term /= 2.0; // or *= 0.5
    counter++;
 }
 System.out.println( "total " + total );
 System.out.println( "number" + counter );
```

“do” loop

Syntax:

```
[initialization]
do {
   [statements]
   [iteration]
} while ( boolean-expression )
```



“do” loops

```
int number = 1;

do {
     ....;
} while ( ++number <= 10 );
```

Examples of "do" loops

- To read in positive numbers until a zero is encountered, and print the biggest one.

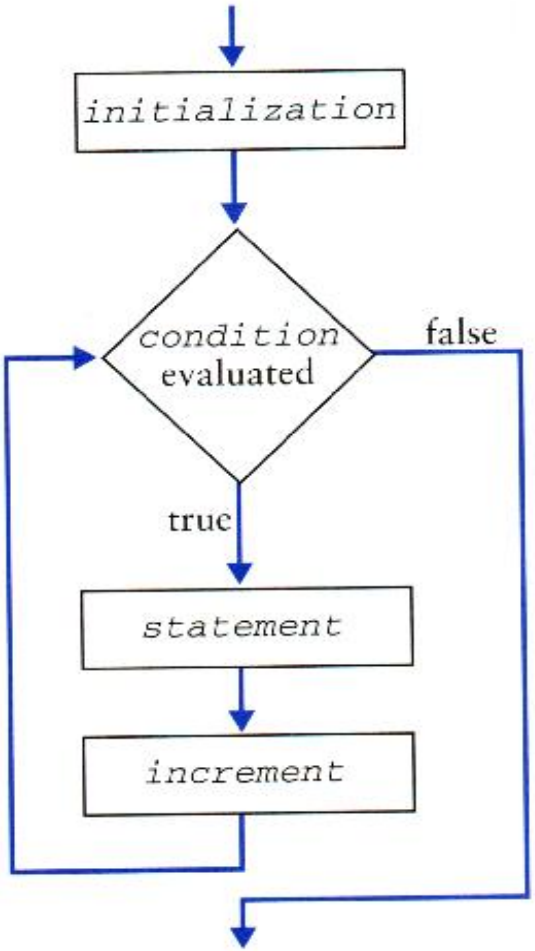
```
int nextnumber, biggest = 0;
do {
    nextnumber = G51OOPInput.readInt();
    if ( biggest < nextnumber ) {
       biggest = nextnumber;
    }
} while ( nextnumber != 0 );
System.out.println( "biggest " + biggest );
```

Using `System.exit()`

- We may wish to abandon the program from within the body of the loop if some error condition occurs.

```
int nextnumber;
do {
    nextnumber = G51OOPInput.readInt();
    if ( nextnumber < 0 ) {
       System.out.println( "Error, negative number" );
       System.out.println( "Value " + nextnumber );
       System.exit( -1 );
    }
    .. process the number ..
    .. which must be >= 0 ..
} while ( nextnumber > 0 );
```

"for" loops



Syntax:

```
for ([ initialization ];
     [ boolean-expression ];
     [iteration]) {
        [statements]
     }
```

Examples :

```
for ( num = 10; num > 0; num-- )
{
   ....;
}
for ( count = 0; count < 10; count++ )
{
   ....;
}
```

"for" loops

```
final float DELTA = 0.00001;
float term;
for ( term = 1.0; term > DELTA; term *= 0.5 ) {
   ....;
}
```

- It is common to declare the loop variable at the start of the for loop itself :

```
for (int count = 0; count < LOOP_COUNT; count++) {
   ....;
}
```

- The general form of a "for" loop is :

```
for ( initialise; test; execute after loop ) {
   ....;
}
```

"for" loops - Readability

- One of the important advantages of a "for" loop is its readability. All of the essential loop control is grouped together at the top of the loop. We can see at a glance the initial values which are set up, the test to be satisfied for loop exit, and the main variable increments. You should make maximum use of this readability.

- The "for" loop could be written as a "while" loop in the form :

```
declaration;
initialise;
....
while ( test ) {
    ....;
    incr;
}
```

- In this layout, the loop control is not so clearly seen.

Defaults

- Defaults are fairly obvious;
- any or all of the three control statements can be omitted. The construct

```
for ( ; ; ) {
    ....;
}
```

- gives no initialisation, assumes a TRUE test result, and performs no incrementing.
- You may find the comma "operator" useful in the initialisation and increment parts of the loop control.

```
for (
    this = 10, that = 0;
    this > that;
    this--, that++
) {
    ....;
}
```

for – each loop

- The basic *for* loop was extended in Java 5 to make iteration over arrays and other collections more convenient. This newer *for* statement is called the *enhanced for* or *for-each* (because it is called this in other programming languages). I've also heard it called the *for-in* loop.
- **Use it** in preference to the standard for loop if applicable because it's much more readable.
- **Series of values**. The *for-each* loop is used to access each successive value in a collection of values.
- **Arrays and Collections**. It's commonly used to iterate over an array or a Collections class (eg, ArrayList).

The "`break`" statement

- In any of the above loops, the special statement "`break`" causes the loop to be abandoned, and execution continues following the closing curly brace.

```
 while ( i > 0 ) {
      ....;
      if ( j == .... ) {
          break; // abandon the loop
      }
      ....;
     }
System.out.println( "continues here ...");
```

- The program continues after the end of the loop.
- Within a nested loop, "break" causes the inner most loop to be abandoned.

The "`continue`" statement

- In any of the above loops, the statement "`continue`" causes the rest of the current round of the loop to be skipped, and a "`while`" or "`do`" loop moves directly to the next test at the head or foot of the loop, respectively;
- a "`for`" loop moves to the increment expression, and then to the test.
- Note that labeled `break` and `continue` are available but beyond the scope of this course and not usually good practice.

Decreasing powers of 2

- To print (in decimal) the decreasing powers of 2 (1, 1/2, 1/4, 1/8, ...) you would write:

```
int count = 1;
final float LIMIT = 0.00001f;
for ( float x = 1.0f; x > LIMIT; x *= 0.5f,count++ ) {
  System.out.println( "count " + count + ", x " + x );
}
```

```
count 1, x 1.0
count 2, x 0.5
count 3, x 0.25
count 4, x 0.125
count 5, x 0.0625
count 6, x 0.03125
count 7, x 0.015625
count 8, x 0.0078125
count 9, x 0.00390625
count 10, x 0.001953125
count 11, x 9.765625E-4
count 12, x 4.8828125E-4
count 13, x 2.4414062E-4
count 14, x 1.2207031E-4
count 15, x 6.1035156E-5
count 16, x 3.0517578E-5
count 17, x 1.5258789E-5
```