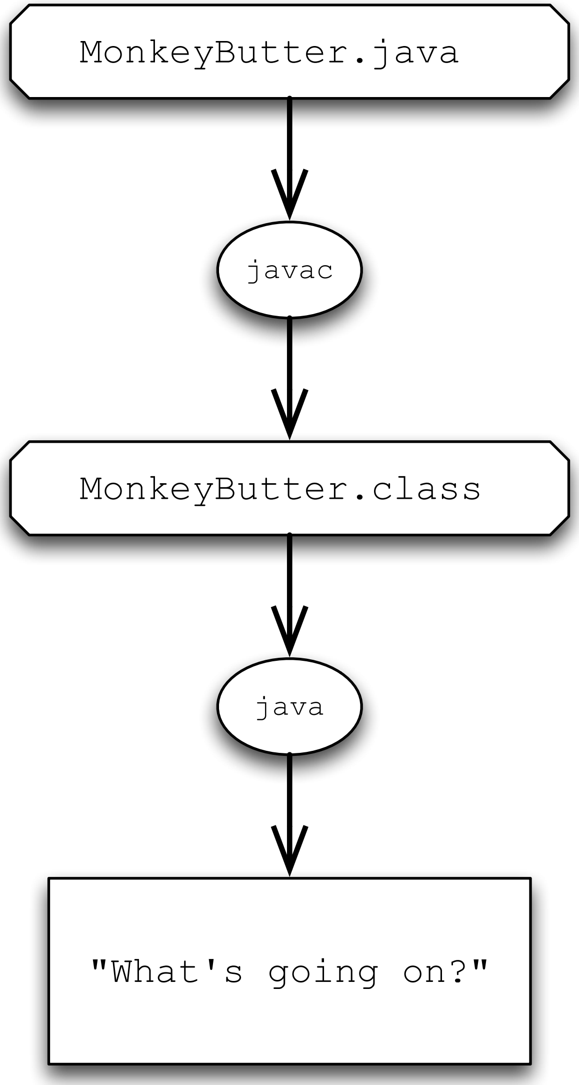
Java cheat sheet for CS 1300

This file will be updated so check back.

A Java source file named "MonkeyButter.java" must contain a class called "MonkeyButter". Likewise, if your class is called "MonkeyButter", it better be in a file called "MonkeyButter.java".

```
class MonkeyButter {
    public static void main(String[] args) {
        System.out.println("What's going on?");
    }
}
```

Java uses semicolons to end (most kinds of) statements. It uses curly braces to specify a block of code. Like Python, blocks may be nested inside one another (as is the case above).



MonkeyButter.java

javac

MonkeyButter.class

java

"What's going on?"

This is your source file.

'javac' is the Java compiler.

This is the Java byte code file.

'java' is the Java virtual machine. It runs your program by finding the 'main' function and begins execution there.

Java has many built-in data types that all other things are based on.

Type	Description	Example Literals
int	Integers	0, 52, -32
float	Floating Point Numbers	0.0f, 52.3f, (float)-32
double	Double-precision floats	0.0, 52.3, -32d
String	Character sequences	"", "Howdy!"
char	Individual characters	'c', 'x', '\n'
boolean	True/False values	true, false

You have to declare variables before (or at the same time) that you use them. This includes the *type* and the variable *name*:

```
int x;
x = -10;
float y = 47f;
String msg = "Sum:";
double sum = (double) (x + y);
boolean is_positive = sum > 0;
```

Declare an int called x

Then use the variable by giving it a value.

Strings are made from double quotes in Java.
Character literals use single quotes.

```
String month = "January";
char grade = 'A';
String gradeAsString = "A";
```

Math operations are similar to every other language on the planet.
Note that Java gives access to many functions in the 'Math' class.

Addition	x + y
Subtraction	x - y
Multiplication	x * y
Division	x / y
Modulus	x % y
Parenthetic Grouping	(x / (y * 3)) - 4
Square Root	Math.sqrt(x)
Exponentiation	Math.pow(x, y)
Sine	Math.sin(x)
Cosine	Math.cos(x)

Boolean Math

Equality	x == y
Inequality	x != y
Greater Than	x > y
G.T. or Equal	x >= y
Less Than	x < y
L.T. or Equal	x <= y
And	x && y
Or	x || y

If statements

```
if (x == 3) {
   // code
} else if (x < 10) {
   // code
} else if (x >= 100) {
   // code
} else {
   // last resort code
}
```

While statements

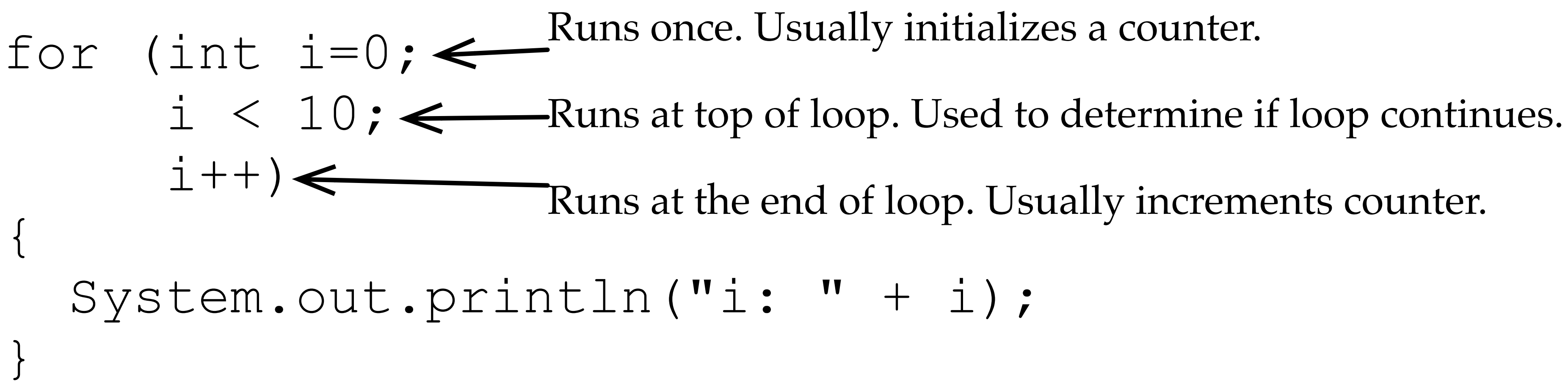
```
while (x > 4.0) {
  x = x * 0.8;
}

while (x && y || z) {
  // code
}
```

For statements (first kind)

```
for (int i=0; i < 10; i++) {
  System.out.println("i: " + i);
}
```

Inside the parentheses, there are three special slots separated by semicolons. The first runs one time at the beginning of the for-loop. The second runs every time at the *top* of the loop. The last runs every time at the *end* of the loop. Here's the same for loop broken up into several lines:



```
for (int i=0;
     i < 10;
     i++)
{
  System.out.println("i: " + i);
}
```

Runs once. Usually initializes a counter.

Runs at top of loop. Used to determine if loop continues.

Runs at the end of loop. Usually increments counter.

For statements (second kind)

```
for (String s : bunchOfStrings) {
  System.out.println("s: " + s);
}
```

This form of the *for* loop lets you iterate through some collection of items. This is the same idea as Python's *for*-loop. In this version you don't get a counter; in the other version you do. Which one you use depends on the situation.

The collection you iterate over can be an Array, or one of the proper collection types in `java.util`, or anything else that can be iterated over.

Converting Types can be straightforward or non-obvious. Given what you have, and what you want, here's what to do:

Have:	**Want:**	**Do This:**
`String x`	`int`	`Integer.parseInt(x)`
	`double`	`Double.parseDouble(x)`
	`boolean`	`Boolean.parseBoolean(x)`
`int x`	`String`	`"" + x`
	`double`	`(double) x`
	`float`	`(float) x`
`float x`	`String`	`"" + x`
	`double`	`(double) x`
	`int`	`(int) x`
`Object x`	`String`	`"" + x`
	`String`	`x.toString()`

You can convert to a String with the + operator. If you have a numeric type and want a different numeric type you can cast it using `(int)`, `(float)`, `(double)`, etc.

Arrays

```
String[] vals = { "Foo", "Bar", "Baz" };
System.out.println(vals[0]);      → Foo
System.out.println(vals[1]);      → Bar
System.out.println(vals.length); → 3
```

Another way to initialize:

```
String[] vals = new String[3];
vals[2] = "Last one";
int n = 5;
String[] moreVals = new String[n];
moreVals[3] = "Penultimate";
```

Use a literal integer to size the array.

Or a non-negative integer variable.

Arrays give you random access with an integer index.

Everything in Java is defined inside a class. This is a big difference from Python and C++. So if you have a function, it is either a member function (a.k.a. an object method), or it is a static function associated with a class.

```
public class SomeFunctions {

  // 'sum' is a member variable that instances
  // have access to, but the class does not.
  int sum; // no 'static' keyword --> member var.

  // this is associated with the class because
  // of the 'static' keyword:
  public static String getHello() {
      return "Hello";
  }

  // this is associated with instances of the
  // SomeFunctions class because it lacks 'static'.
  public int addToMemberVar(int addme) {
    sum = sum + addme;
    return sum;
  }
}
```

Here's how you create objects and make method calls. Notice that the *instance* called 'sf' used used for the regular non-static method 'addToMemberVar', but we use the class name 'SomeFunctions' to use the static function 'getHello'.

```
SomeFunctions sf = new SomeFunctions();

// use the instance method
System.out.println(sf.addToMemberVar(42));
System.out.println(sf.addToMemberVar(8));

// use the static class function
System.out.println(SomeFunctions.getHello());
```