Control Structures

If statements and while loops

What we know so far

Java lets us declare variables of different types.

Declaring the type of variable tells Java how much storage space it will need, and also allows the compiler to automatically check some aspects of our code for correctness.

Variables can be used in expressions involving operators.

Operators allow us to do arithmetic, boolean logic, compare expressions, and assign values to variables.

Java executes the statements in the main method, in order from top to bottom.

Conditional execution

Up until now, we've written our programs to do exactly and only what we wanted. Changing the behavior of the program means changing the code, and changing the code means recompiling.

But wouldn't if be nice if we could write code that could behave differently under different circumstances?

This is the idea that leads to our first control statement: **`if`**

If... then?

The structure works like this:

```
int x = 4;
if( x > 2 )
{
  System.out.println( "That is a nice x" );
}
```

The value of **`x`** is used to determine if the block of code below the **`if`** statement will run or not.

In general

More abstractly, we have:

```
if( condition )
{
  // controlled code here
  // only runs if condition is true
}
```

The condition must be a statement that gives a boolean result. If that result is true, then the block containing the controlled code is executed. If not, execution continues immediately below the controlled block.

Note of caution

You will not and should not find a semicolon after the parenthesis. Really. Seriously:

```
if( condition )
{
  // controlled code here
}
```

The statement is considered to end when the controlled block ends. So the whole thing is one statement, although people often say "if statement" when they mean the first line up there. In any case, because the statement ends with a block, it doesn't need a semicolon. No semicolon. OK.

So that's nice...

But what if we wanted to handle both cases of a condition? We could always use multiple if statements:

```
if( x > 2 )
{
  System.out.println( "That is a nice x" );
}
if( x <= 2 )
{
  System.out.println( "That's OK too" );
}
```

Wait, that's annoying

No one wanted to do the work of figuring out the opposite condition and testing for that too. So we invented this:

```
if( x > 2 )
{
  System.out.println( "That is a nice x" );
}
else
{
  System.out.println( "That's OK too" );
}
```

More formally

```
if( condition )
{
  // this code runs if condition is true
}
else
{
  // this code runs if condition is false
}
```

So then we decided to get fancy...

```
if( x > 2 )
{
  System.out.println( "That is a nice x" );
}
else
{
  if( x > 0 )
  {
    System.out.println( "That's positive" );
  }
  else
  {
    System.out.println( "Thanks anyway" );
  }
}
```

And that was just fine.

The blocks of code controlled by if statements or if-else statements are just like any other block of code. They can contain any statements, including other if statements.

There is no limit to how many if statements can be nested inside other if statements.

But in our previous example, we were really just trying to handle three different cases with regard to the value of x.

But isn't this nicer?

```
if( x > 2 )
{
  System.out.println( "That is a nice x" );
}
else if( x > 0 )
{
  System.out.println( "That's positive" );
}
else
{
  System.out.println( "Thanks anyway" );
}
```

More formally

```
if( condition1 )
{
  // this code runs if condition1 is true
}
else if( condition2 )
{
  // if condition1 is false, then we test
  // condition2. if true, this code runs.
}
else
{
  // this code runs if both are false
}
```

If-elseif-else

You can have as many `else if` blocks as you want. Each condition will be checked until one is found to be true. If none are true, the `else` block will be run (if it exists).

Note that `else if` and `else` blocks can only appear under an `if` statement. The `else if` and `else` blocks represent something to do when the first `if` condition is false. So having them alone makes no sense.

Repeated Execution

Up until now, we've written our programs sequentially. We just learned how to make Java decide at run time if it should skip over sections of code (with the `if` statement) or choose between different blocks (by combining `if` and `else`).

However, if you want to see 10 lines of output, you still have to write 10 `System.out.println` statements.

Wouldn't it be nice to tell Java to repeat something?

The while loop

```
int x = 0;
while( x < 10 )
{
  System.out.println( "Counting... " + x );
  x++;
}
```

And this does just what you'd hope: it prints ten lines of output.

Formally

```
while( condition )
{
  // controlled code here
}
```

The difference between a while loop and an if statement is that when we finish executing the controlled block of a while loop, Java will return to check the condition again.

If the condition is still true, it executes the controlled block again. This means that we need to change something at some point to make the condition false. If we don't, we will be caught in an infinite loop.