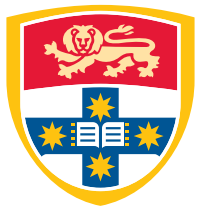
INFO1103: Introduction to Programming

School of Information Technologies, University of Sydney



Lecture 4: Expressions, Command-line arguments, Conventions, and Pseudocode

Manipulating information, Immediate input to your program; Style; how to write algorithms

Using the command-line

Getting information in to your program

Reading values in with Scanner

Recall that we can use a Scanner to read in values to make an interactive program:

```
1  import java.util.Scanner;
2  public class ReadADouble {
3      public static void main(String[] args) {
4          System.out.println("Enter your height in metres: ");
5          double height; // this is DECLARING the variable;
6          Scanner keyboard = new Scanner(System.in);
7          height = keyboard.nextDouble(); // ASSIGNING the variable
8          System.out.println("You entered " + height + "m.");
9          System.out.println("If you were 10% bigger you'd be "
10             + (height*1.1) + "m.");
11         keyboard.close();
12     }
13 }
```

Reading values in with Scanner (cont.)

```
~> javac ReadADouble.java
~> java ReadADouble
Enter your height in metres:
1.82
You entered 1.82m.
If you were 10% bigger you'd be 2.0020000000000002m.
~>
```

Using the command-line arguments

If you don't want to use the `Scanner`, such as if you want to run your program like this:

```
~> java GrowMe 1.82
You entered a height of 1.82m.
If you grow by 10% then you'll be 2.002m.
```

. . . then you have to use the *command-line arguments*

The command-line arguments

When you run a java program from the command-line or terminal you do something like this:

```
> java HelloWorld
Hello, World!
```

. . . but you can also give a program information directly, as soon as you call it, like this:

```
> java Sum 4 5.5
The sum of 4 and 5.5 is 9.5
```

How does this work?

main and String[] args

The main program always begins something like this:

```
1       public static void main(String[] args)
```

The variable args gives you a way to access the pieces of information you provide, like 4 and 5.5 above.
The way we do that is by referring to them using the square brackets, also called the "index operator", like this:

```
1  public class Sum {
2     public static void main(String[] args) {
3        double x = Double.parseDouble(args[0]);
4                                      ^^^^^^^ first argument
5        double y = Double.parseDouble(args[1]);
6                                      ^^^^^^^ second argument
7        System.out.println("The sum of " + x + " and " + y
8              + " is " + (x+y));
9     }
10 }
```

The arguments are always Strings: they have to be converted into numbers if you want to use them as numbers.

Double.parseDouble

`Double.parseDouble` is a way of converting a `String` into a `double`.

This is extremely useful if you need to read in floating-point numbers to a program when you run it.

Use it like this:

```
1       double x = Double.parseDouble("12.3");
```

now the String "12.3" has been *parsed* — that is, *read and understood* — as a `double` value, and then stored in `x`.
The original `String` is left untouched: it doesn't change.

(There is also `Integer.parseInt` : it parses a string like "123" as the integer number 123.)

Expressions

How to do calculations

Simple expressions

An *expression* is just a combination of variables and other items that can be *evaluated*, and will have a *value* like `true`, `false`, 25, 1.4142, etc.

Here are some expressions:

4	A simple number;
$x + y$	A mathematical formula;
$\sqrt{3}$	Another one;
$(A == B)$	A Boolean expression, which will evaluate to either true if A equals B, or false otherwise

Assignment

We *assign* values to variables using a single equals sign, like this:

```
1        int x = 4;
2        int y = x;
```

After this, y has the value 4.

```
1        x = 2*y;
```

Now x has the value 8, and y still has the value 4.

We construct expressions using *operators*

An operator is a symbol or small set of symbols that let you perform some kind of operation on the *operands*.
Here are some:

`x = 5`	*assigns* the value of `x` to be 5
`a + b`	*adds* `a` and `b`
`!sad`	*negates* the Boolean value of the variable `sad`
`x++`	adds 1 to the variable `x`: this is called *incrementing*
`a < 23`	*compares* the value of `a` to 23

Operator terminology

Operators work on *operands*. They may or may not modify the operand.

An operator can be

unary	operating on one operand;
binary	operating on two operands
ternary	operating on three operands

Assignment operator =

There are several operators in Java to make your life easier. The most common you'll probably use is the *assignment* operator:

=

We've seen this before: use = to assign the value on the left to take the value on the right:

$$lvalue \leftarrow rvalue$$

$$lvalue = rvalue$$

And remember the equality operator is ==: it is used to compare whether two *primitive types* are equal.

Operators +, -, *, /

The next set of operators are very straightforward: they are the standard *operators* of mathematics:

+	−	×	÷	(mathematical)
+	-	*	/	(code)

For example:

```
1       int x = 5;
2       int y = 3;
3       int z = x + y;   // z gets the value of (x+y), which is 8
4       x = -x;          // now x is -5
5       z = z + x;       // now z is 3
6       y = y * 2;       // now y is 6
```

Warnings about operators

Integer division

```
1    int x = 3 / 2;
2    System.out.println(x);
3
4    int y = 1 / 0;
5    System.out.println(y);
```

Embedding shorthand operators is dangerous.
What do you think happens in the following?

```
1    int x = 4;
2    boolean matches = (x-- == 3);
3    int y = (x += 7) + ( --x );
```

This is how bugs get introduced. This needs to be readable. It should be expanded to show each calculation in a separate statement.

```
x = 9
y = 19
```

Simple Calculations

```
1  public class SimpleOperators {
2      public static void main(String[] args) {
3          int x = 6;
4          int y = x++;
5          int z = (x + --y);
6          int n = 0;
7          n++;
8          System.out.println("x = " + x);
9          System.out.println("y = " + y);
10         System.out.println("z = " + z);
11         System.out.println("n = " + n);
12         x = 5;
13         float r = 1.2f; // the 'f' means treat it as a float
14         float s = x * r;
15         System.out.println("x = " + x);
16         System.out.println("r = " + r);
17         System.out.println("s = " + s);
18         s = s - 1.1f;
19         System.out.println("s = " + s);
20     }
21 }
```

Simple Calculations

Let's just look at the relevant bits:

```
6         int x = 6;
7         int y = x++;
8         int z = (x + --y);
9         int n = 0;
10        n++;
11        System.out.println("x = " + x);
12        System.out.println("y = " + y);
13        System.out.println("z = " + z);
14        System.out.println("n = " + n);
15        x = 5;
16        float r = 1.2f; // the 'f' means treat it as a float
17        float s = x * r;
18        System.out.println("x = " + x);
19        System.out.println("r = " + r);
20        System.out.println("s = " + s);
21        s = s - 1.1f;
22        System.out.println("s = " + s);
```

What does this print?

Simple Calculations (cont.)

```
~> javac SimpleOperators.java
~> java SimpleOperators
x = 7
y = 5
z = 12
n = 1
x = 5
r = 1.2
s = 6.0
s = 4.9
```

+ and Strings

Above, in the lines

```
10          System.out.println("z = " + z);
11          System.out.println("n = " + n);
12          x = 5;
```

and

```
14          float s = x * r;
```

there is a "+" sign in arguments of the `println` method call. It's there to *concatenate* two `String`s.
`"x = "` is a `String`, and `println` can only print `String` objects, so it converts the whole expression to a single `String`.
What do you think this prints out:

```
1       System.out.println("5" + "3");
2       System.out.println(5 + 3);
```

In general you can use the “+” to concatenate any two `Strings`, e.g., like this:

```
1    String msg = "Hello, " + args[0] + ", how are you?";
```

Operators +=, -=, *= and /=

These operators are a very nice shorthand. They all operate in the same way, by modifying the operand on the left, using the operand on the right.

shorthand	equivalent to
`x += n;`	`x = x + n;`
`x -= n;`	`x = x - n;`
`x *= k;`	`x = x*k;`
`x /= k;`	`x = x/k;`

In general,

`x` □`= y`

is equivalent to

`x = x` □ `y`,

for whatever □ is.

Equality operator ==

This binary operator should be very familiar by now: use == to return the value *true* when the two operands are equal:

left value == right value

is true if and only if the two values are the same.

Comparing ints and booleans

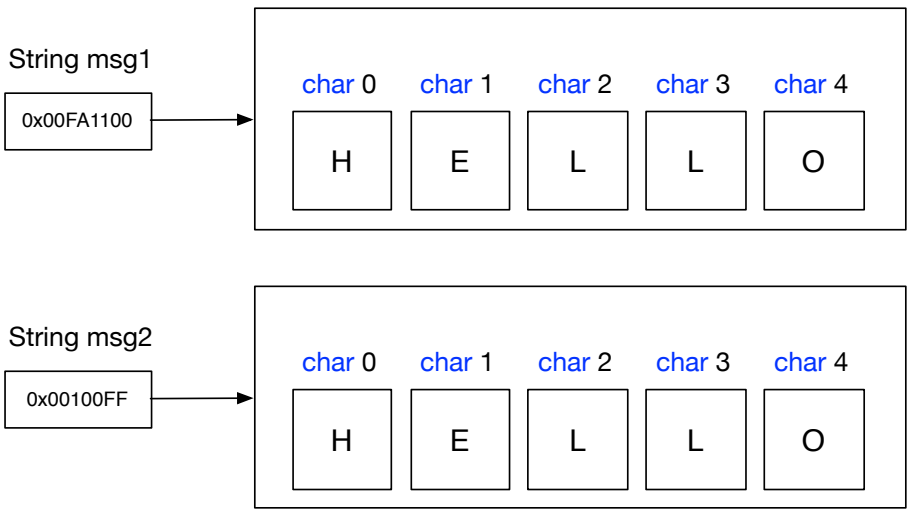
```
1    int x = 4;
2    int y = 4;
3    int z = 2;
4    boolean xySame = (x == y);
5    boolean xzSame = (x == z);
6    System.out.println("xySame = " + xySame);
7    System.out.println("xzSame = " + xzSame);
8    System.out.println("(xySame == xzSame) = " + (xySame ==  xzSame));
```

```
> emacs Equality.java
> javac Equality.java
> java Equality
xySame = true
xzSame = false
(xySame == xzSame) = false
```

Comparing `Strings`: *str2 == str2 ?*

Don't use == to compare String content

== compares values. The value of the String variable is a *memory address*.
This has no information about the content of that area of memory.
The equals() method will visit the area of memory of two strings and compare the content.




```
1    String msg1 = "hello";
2    String msg2 = "hello";
3    if (s.equals(s2)) {
4        // ...
5    }
```

not!

In Java the negation operator looks like this: "`!`" — it's the exclamation mark. In code you'll see this quite often, for example in expressions like

```
if (x != y),
```

which is true if `x` is *not* equal to `y`, or like this:

```
if (!(x == y)),
```

which is true if the statement "`x == y`" is *false*.

In mathematical symbols we use the "negate" symbol ¬ for "not".

- The value of $\neg x_1$ is true if, and only if, x_1 is *false*.

Parentheses

I've been using expressions inside parentheses `(`, `)`.
Expressions, when they are evaluated/executed, have a *value*.
That means if I write something like

```
1    int x = 4;
2    int y = 4;
```

then if the expression `(x == y)` is executed, it will give the result "`true`".
Writing the expression in parentheses is usually a good idea. It avoids confusion!

`!(x == y)` is equivalent to `(x != y)` but `(!x == y)` is different

And and Or

Let's think about a set of variables, called $x_1, \ldots x_k$.

- The value of (x_1 AND x_2) is true if, and only if, both x_1 and x_2 are true.
- The value of (x_1 AND x_2 AND $\ldots$AND x_k) is true if, and only if, all of the x_i are true.
- The value of (x_1 OR x_2) is true if, and only if, at least one of x_1 and x_2 is true.
- The value of (x_1 OR x_2 OR $\ldots$OR x_k) is true if, and only if, at least one of the x_i is true.

In Java we write `&&` for logical AND, and `||` for logical OR.

Operator Precedence

What is the result of:

```
1        int x = 1 + 5 * 3 + 2 ;
2        int y = 8 / 4 - 1 / 1 - 1;
```

This particular programming language has this order: Brackets, Operators, Division / Multiplication, Addition / Subtraction (BODMAS)

Always *use parentheses* to be clear.

```
1        int x = (1 + 5) * (3 + 2) ;
2        int y = 8 / (4 - (1 / 1) - 1);
```

Assignment

In Java, if we want to set the value of something we use *assignment* that looks like this:

```
3       float x = -1/2;
```

Just remember the value on the *left* gets the value of the expression on the *right*.
In pseudocode we'd write $x \leftarrow \frac{-1}{2}$: the left-arrow is often called "gets" to mean the variable on the left *gets* the value on the right.

Assignment is not equality

Saying $x = 3$ in mathematics means "x is a variable whose value is currently 3".
In that sense the equivalent statement is $3 = x$, but writing that in a Java program doesn't make sense: you would be attempting to change the value of 3!

If you want to test whether x has the value 3, you would evaluate the following expression:

```
1        (x == 3);
```

which has the Boolean value *true* if x really does equal 3 and the value *false* otherwise.

Don't confuse '=' (assignment) with '==' (equality comparison)!

Comparison — warning!

Comparing floating point numbers for equality is a BAD IDEA.

floating point numbers are stored to finite precision. $\frac{1}{3}$ is not stored exactly, but approximated.

```
1  public class Rounding {
2     public static void main(String [] args) {
3          float f1 = 0.00015f + 0.00015f;
4          float f2 = 0.0002f + 0.0001f;
5          boolean matches = ( (f1 == f2) );
6          System.out.println("f1 = " + f1 );
7          System.out.println("f2 = " + f2 );
8          System.out.println("matches = " + matches);
9     }
10 }
```

prints out

```
f1 = 3.0E-4
f2 = 2.9999999E-4
matches = false
```

This is a serious problem.

That's all, folks!

This is the end of the lecture material covered in Week 2.