

There are development environments, like CodeWarrior, JBuilder, JCreator and Eclipse, which provide an integrated development environment for building and debugging programs. They vary from expensive to free, with cost not being a guide to quality or ease of use!

You can get Java for free from Sun as follows...

↳ <http://java.sun.com>

↳ J2SE (Core/Desktop)

↳ go to RH pane "Related Links"

"Popular Downloads" → J2SE 1.4.2

↳ J2SE v 1.4.2_04 SDK ... Download J2SE SDK

After you've installed this, if you have space it's worth downloading the documentation J2SE v1.4.2 Documentation which arrives as a zip file.

Also on the sun site (LH pane "Learning" Tutorials & Code Camps) is Tutorials which takes you to a great sequence "Java Series" of easy and well-written tutorials ... start with The Java Tutorial which leads you through much of what's in this course starting with installing the software!

A widely used (and free) powerful IDE can be found at

↳ www.eclipse.org

↳ "What's New" June 25 ... download Eclipse Project release 3

Your chosen site should grab eclipse-SDK-3.0-win32.zip (or a Mac or Unix version depending on your computer). You don't need an IDE to run Java, but it's sometimes nice when things get complicated.

Primitive Types

byte	$-128 \leq \text{integer} < 127$
short	$-32768 \leq \text{ " } < 32767$
int	$-2^{31} \leq \text{ " } < 2^{31} - 1$
long	$-2^{63} \leq \text{ " } < 2^{63} - 1$
float	$\pm 10^{-46} \leq \text{decimal} < 10^{38}$
double	$\pm 10^{-324} \leq \text{ " } < 10^{308}$
char	single unicode character
boolean	false, true

The declaration

```
int boo;
```

makes **boo** an allowable name for an integer. The declaration and initialization

```
int boo = 13074;
```

makes **boo** an allowable name for an integer, and before it can be used, initializes its value to 13074.

be careful that = is really assignment, not 'equals'.

Similarly, we can have

```
double whoosh = 9.874;
```

```
char cuckoo = 'T';
```

```
boolean ouch = false;
```

note the single quote

Awkward characters like ? or ' can be assigned using \ as in

```
cuckoo = '\?';
```

```
cuckoo = '\';
```

Arithmetic

+	plus	$3 + 4;$	\longrightarrow	7
-	minus	$3 - 4;$	\longrightarrow	-1
*	times	$3 * 4;$	\longrightarrow	12
/	divide	$3 / 4;$	\longrightarrow	0
%	remainder	$3 \% 4;$	\longrightarrow	3

What do you think $(-3) \% 4;$ evaluates to?

[As an aside, it's worth noting that there is a non-primitive type **String** which carries a string of characters, and

```
String tut = "Methinks I were,";
```

```
String um = " there is no man...";
```

```
tut = tut + um;
```

gives tut the updated value of

Methinks I were, there is no man...

[So for strings, + appends the second string to the end of the first string.

Also, for those who like calculating...

```
Math.sin(1.78);
```

```
Math.atan(72.4); etc
```

all do the obvious — **Math** is a repository of lots of useful stuff!

Now for an example . . .

```
import java.io.*;
```

```
public class Multiplier  
{
```

```
    public static void main (String [] args) throws Exception  
    {
```

setting
up the
I/O

```
        InputStreamReader isr = new InputStreamReader (System.in);  
        BufferedReader comingIn = new BufferedReader (isr);  
        PrintWriter goingOut = new PrintWriter (System.out, true);
```

```
        int x, y; ← space to store input  
        long z = 0; ← bigger space for answer !  
        String ask = "Please enter an integer.";
```

get the
first number

```
        goingOut.println (ask);  
        x = Integer.parseInt (comingIn.readLine());
```

get second
number

```
        goingOut.println (ask); ← if what was 'read' can't  
        y = Integer.parseInt (comingIn.readLine()); ← be an integer, throw Except
```

```
        z = x * y; ← actually perform the  
                    multiplication !!
```

give out
the answer

```
        goingOut.print ("The value of " + x + " times " + y);  
        goingOut.println (" is " + z);
```

unremitting
courtesy !

```
        goingOut.println ("Thanks for using \"Multiplier!\");  
        goingOut.println ("Do come again!");  
        comingIn.close();
```

```
    } // end of class Multiplier
```

There are also various 'shorthands' ...

$a = a + 12;$
 $a = a - 4;$
 $a = a * 3;$
 $a = a / 5;$

$a++$	$a = a + 1$
$a--$	$a = a - 1$

$int\ a, b;$

$a = 17;$

$a += 12;$ $\rightarrow a \rightarrow 29$

$a -= 4;$ $\rightarrow a \rightarrow 25$

$a *= 3;$ $\rightarrow a \rightarrow 75$

$a /= 5;$ $\rightarrow a \rightarrow 15$

$b = 2 * (a++);$ $\rightarrow (a \rightarrow 16)$
 $(b \rightarrow 30)$

$a = 4 * (++b);$ $\rightarrow (a \rightarrow 124)$
 $(b \rightarrow 31)$

$a--;$ $\rightarrow a \rightarrow 123$

$--b;$ $\rightarrow b \rightarrow 30$

Type conversion is also very handy ...

$int\ a = 73, b = 10;$

$double\ c;$

$c = a / b;$ $\rightarrow c \rightarrow 7$

$c = \underline{(double)\ a / b};$ $\rightarrow c \rightarrow 7.3$

precedence

\rightarrow anon double variable

Comparisons

$a == b$

$a < b$

$a > b$

$a != b$

$a <= b$

$a >= b$

These have the obvious meanings for the primitive types.

Logic

A && B	AND	A & B
A B	OR	A B
!A	NOT	

So for example, $A \neq B$ and $!(A == B)$ are equivalent.

The difference between $\&$ and $\&\&$ (similarly for $|$ and $||$) relies on "short-circuiting".

$(3 == 7) \&\& (2 == 3/0)$

evaluates comfortably to **false**, since failure occurred in the first term there was no need to go further; but

$(3 == 7) \& (2 == 3/0)$

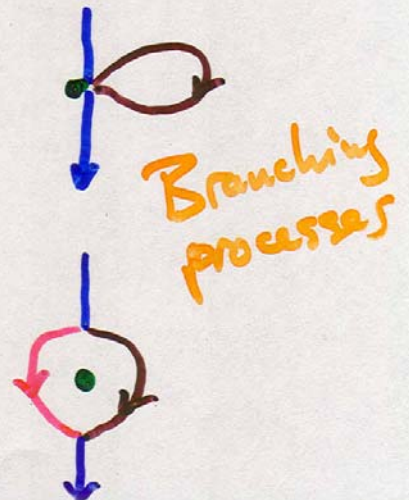
is a disaster, since the single ampersand has no short-circuit provision.

Control

```
if (true)
{
}
```

and

```
if (true)
{
}
else
{
}
```



```
while ( xxxx )  
  { xxxxxx }
```



and

```
do  
  { xxxxxx }  
while ( xxxx );
```



and

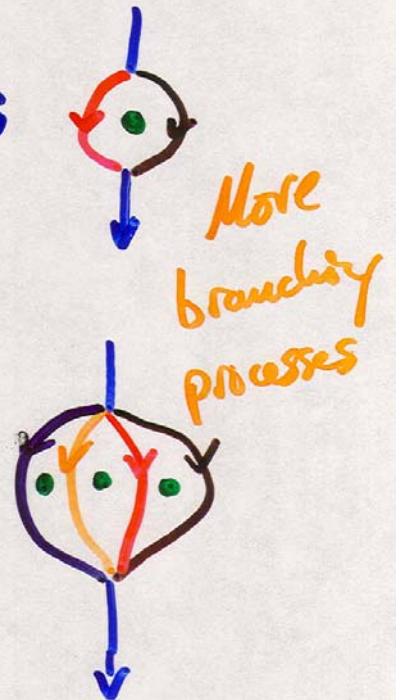
```
for ( xxxx; xxxx; xxxx )  
  { xxxxxx }
```



```
xxxxxx ? xxxxxx : xxxx ;
```

and

```
switch ( name )  
{  
  case value : xxxxxx break ;  
  case value : xxxxxx break ;  
  case value : xxxxxx break ;  
  default : xxxxxx  
}
```



We can now broaden our previous example ...

```
import java.io.*;
```

```
public class Arithmetic  
{
```

```
    public static void main (String [] args) throws Exception  
    {
```

```
        InputStreamReader isr = new InputStreamReader (System.in);
```

```
        BufferedReader comingIn = new BufferedReader (isr);
```

```
        PrintWriter goingOut = new PrintWriter (System.out, true);
```

```
        char op;
```

```
        int x, y;
```

```
        double answer;
```

```
        String ask = "Please enter an ";
```

```
        goingOut.println (ask + "integer.");
```

```
        x = Integer.parseInt (comingIn.readLine());
```

```
        goingOut.println (ask + "operator.");
```

```
        op = (comingIn.readLine()).charAt(0);
```

```
        goingOut.println (ask + "integer.");
```

```
        y = Integer.parseInt (comingIn.readLine());
```

```
        if (op == '+')
```

```
            answer = x + y;
```

```
        else if (op == '-')
```

```
            answer = x - y;
```

```
        else if (op == '*')
```

```
            answer = x * y;
```

```
        else if (op == '/')
```

```
            if (y != 0) answer = x / y;
```

```
            else answer = x / y;
```

```
        goingOut.println ("The answer is " + answer);
```

```
        comingIn.close();
```

```
    }  
}
```

*we should handle
this a bit better*

*this will
create an
exception*