

# *Notes from Well House Consultants*

*These notes are written by Well House Consultants and distributed under their Open Training Notes License. If a copy of this license is not supplied at the end of these notes, please visit*

*<http://www.wellho.net/net/whcotnl.html>  
for details.*

## 1.1 Well House Consultants

Well House Consultants provides niche training, primarily but not exclusively in Open Source programming languages. We offer public courses at our training centre and private courses at your offices. We also make some of our training notes available under our "Open Training Notes" license, such as we're doing in this document here.

## 1.2 Open Training Notes License

With an "Open Training Notes License", for which we make no charge, you're allowed to print, use and distribute these notes provided that you retain the complete and unaltered license agreement with them, including our copyright statement. This means that you can learn from the notes, and have others learn from them too.

You are NOT allowed to charge (directly or indirectly) for the copying or distribution of these notes, nor are you allowed to charge for presentations making any use of them.

## 1.3 Courses presented by the author

If you would like us to attend a course (Java, Perl, Python, PHP, Tcl/Tk, MySQL or Linux) presented by the author of these notes, please see our public course schedule at

<http://www.wellho.net/course/index.html>

If you have a group of 4 or more trainees who require the same course at the same time, it will cost you less to have us run a private course for you. Please visit our onsite training page at

<http://www.wellho.net/course/otc.html>

which will give you details and costing information

## 1.4 Contact Details

Well House Consultants may be found online at

<http://www.wellho.net>

[graham@wellho.net](mailto:graham@wellho.net)

technical contact

[lisa@wellho.net](mailto:lisa@wellho.net)

administration contact

Our full postal address is

404 The Spa

Melksham

Wiltshire

UK SN12 6QL

Phone +44 (0) 1225 708225

Fax +44 (0) 1225 707126

# Variables

Information that's produced by one Java statement and used in later ones is held in variables. In Java, variables must be declared before they are used, and you must specify the type of information they contain. Operations can be performed on variables, and casting can be used to convert information of one type to another.

<i>First use of variables in Java</i> . . . . .	4
<i>Type casting and conversion</i> . . . . .	6
<i>Reading input from the user</i> . . . . .	8

A Java method comprises a number of statements that are performed one after another. Usually, the programmer will wish to take information that's generated as a result of one statement, and make further use of it in a subsequent statement. This is done using a "variable" which is a named 'memory slot' into which a value of a certain type can be saved for later re-use.

## 2.1 First use of variables in Java

In Java, you have to tell the compiler about variables before you use them. This is known as declaring a variable. You also have to tell the compiler what type of information a variable will hold. Java has just eight primitive types built into the JVM itself.

Here's an example using variables:

```
public class Addup {

    public static void main(String [] args) {
        int rovers;
        int city;
        int total;
        rovers = 3;
        city = 2;
        total = rovers + city;
        System.out.print("Today, Bristol teams scored ");
        System.out.print(total);
        System.out.println(" goals");
    }
}
```

```
bash-2.04$ java Addup
Today, Bristol teams scored 5 goals
bash-2.04$
```

Figure 1 Running public class Addup

We have declared three variables, called "rovers", "city" and "total". The = operator tells Java to evaluate the expression to the right and save it in the variable named on the left, so we set rovers to "3" and city to "2". Then we add up the contents of rovers and city, and save the result into total. Finally, we print out our results.

You may have noticed that we used **print** rather than **println** for two out of the three outputs. The **println** method adds a new line on the end of the text it outputs, but the **print** method does not. This allows us to make up a single line of output using multiple **print** statements.

### Variable names

As long as you don't use a reserved word, you can use any variable name you like within the following rules:

- Starts with a letter
  - followed by as few or as many letters, digits and underscores as you like
- Remember that variable names are case sensitive. The variable "rovers" is *not* the same as the variable "Rovers".

We suggest that for variables of this type (primitives), you:

- Use lower case letters throughout
- Make the names descriptive, but not very long

### Declaring and initialising variables

You can save yourself some lines of code if you declare several variables at the same time, initialising them at the same time too. You might also like to note that declarations don't have to be at the top of blocks as they do in some other languages.

```
public class Add2 {

    public static void main(String [] args) {
        int rovers = 1, city = 2;
        System.out.print("Today, Bristol teams scored ");
        int total = rovers + city;
        System.out.print(total);
        System.out.println(" goals");
    }
}
```

Figure 2 Running public class Add2

```
bash-2.04$ java Add2
Today, Bristol teams scored 3 goals
bash-2.04$
```

Variables declared in this way are dynamically created as Java runs. Computer memory is allocated as the declaration is executed, and they stay in existence until the } at the end of the block in which they are declared is reached.

They are not visible outside the block. This means that you can use the same variable name in two or more methods, but if you do, you will have two different variables with the same name.

### Primitive types

"int" means "integer". In other words, a whole number. In Java, `int` variables occupy 4 bytes and so are 32-bit valued. This is a fixed part of the Java specification. Whole number primitive variable types are:

<b>byte</b>	1 byte
<b>short</b>	2 bytes
<b>int</b>	4 bytes
<b>long</b>	8 bytes

If you want to work with numbers with decimal places (i.e. floating point numbers), Java provides:

<b>float</b>	4 bytes
<b>double</b>	8 bytes

There are two further primitives which don't hold numbers at all:

<b>char</b>	to hold a character
<b>boolean</b>	to hold either true or false

That's it! No more primitive types. Most data will be held in objects that are different things, and you can define as many different types of objects as you wish.

## 2.2 Type casting and conversion

Let's say that I want to report the average score of our teams:

```
public class Average {

    public static void main(String [] args) {
        int rovers = 2, city = 3;
        System.out.print("Today, Bristol teams averaged ");
        float total = rovers + city / 2;
        System.out.print(total);
        System.out.println(" goals");
    }
}
```

Figure 3 Running public class Average

```
bash-2.04$ java Average
Today, Bristol teams averaged 3.0 goals
bash-2.04$
```

I've used a / for divide, and I've put the result into a float variable as averages can have decimal parts in them. But the method gives the wrong result, as shown in Figure 3.

Two immediate problems:

- Multiplication and division (\* and /) happen before addition and subtraction.
- Dividing by a whole number causes the remainder to be thrown away.

So  $3 / 2$  gave me 1 (threw away the remainder 1)

The  $2 + 1$  gave me 3

Saved into a float, that became 3.0

Let's attempt to correct that. Round brackets can be used to force the addition to happen first, and we can change 2 into 2.0 to force a floating point division:

```
public class Av2 {

    public static void main(String [] args) {
        int rovers = 2, city = 3;
        System.out.print("Today, Bristol teams averaged ");
        float total = (rovers + city) / 2.0;
        System.out.print(total);
        System.out.println(" goals");
    }
}
```

Figure 4 Running public class Av2

```
bash-2.04$ javac Av2.java
Av2.java:6: possible loss of precision
found   : double
required: float
        float total = (rovers + city) / 2.0;
                                   ^
1 error
bash-2.04$
```

But, alas, that fails to compile, as seen in Figure 4. Java is a fussy language!

The constant 2.0 is actually a double precision constant.<sup>1</sup> If you write a statement that increases the accuracy of a calculation, Java won't complain,<sup>2</sup> but it won't stand for a reduction in accuracy or precision unless you tell it that it's acceptable using a cast.

Let's see two possible solutions then:

```
public class Av3 {

    public static void main(String [] args) {
        int rovers = 2, city = 3;
        System.out.print("Today, Bristol teams averaged ");
        float total = (rovers + city) / 2.0F;
        System.out.print(total);
        System.out.println(" goals");
    }

}

public class Av3b {

    public static void main(String [] args) {
        int rovers = 2, city = 3;
        System.out.print("Today, Bristol teams averaged ");
        float total = (float)((rovers + city) / 2.0);
        System.out.print(total);
        System.out.println(" goals");
    }

}
```

Figure 5 Running public classes Av3 and Av3b

```
bash-2.04$ java Av3
Today, Bristol teams averaged 2.5 goals
bash-2.04$ java Av3b
Today, Bristol teams averaged 2.5 goals
bash-2.04$
```

And they both work (see Figure 5).

A third possible solution: I could have declared total to be a double.

<sup>1</sup> you would write 2.0F if you wanted to force it to be a float

<sup>2</sup> so I was able to take my two integers "rovers" and "city" and turn them into a double before I did the divide

### 2.3 Reading input from the user

Before we finish this module, would you like to write a program that asks the user to enter some data, does a calculation on what he enters, and prints out the result? For sure you would, but you don't know how to read input yet! Although it's a fundamental requirement of most applications to read data, it uses some non-basic concepts of Java. Data input is very prone to errors; users may enter letters rather than numbers, the network connection may go wrong, and a whole host of other things could happen that need trapping and handling – not something that's practical for us to do this early in your learning.<sup>1</sup>

Therefore, we're going to take a different approach. Java is all about using code that other people have written, either as part of the team, or more generally. Such code is put into classes which are then available to you (that is, if they're declared public). To get you into the good practice of re-using other people's code rather than redoing a job that's been done many times before, we're providing you with a class called "WellHouseInput" that will read user inputs from the keyboard.

I would hope that most class providers would give you documentation, but if they don't, you can use the `javap` utility to get a description of the API at least ;-)

```
bash-2.04$ javap WellHouseInput
Compiled from WellHouseInput.java
public class WellHouseInput extends java.lang.Object {
    public WellHouseInput();
    public static float readNumber();
    public static java.lang.String readLine();
}
bash-2.04$
```

Please beware these methods are *not* part of the standard Java distribution, but you're welcome to use them in your own programs if they do what you want. It's up to you to check that's the case!

Let's write a program to read in an quantity and a unit price, and report on the grand total to be paid:

```
public class Cost {

    public static void main(String [] args) {

        // Prompt - read sequences ( x 2) for inputs

        System.out.print("How much do they cost each? ");
        float each = WellHouseInput.readNumber();
        System.out.print("How many do you want to buy? ");
        float quantity = WellHouseInput.readNumber();

        // Do the calculations

        float total = each * quantity;
        System.out.print("Total cost ");
        System.out.print(total);
        System.out.println(" excluding tax and shipping");
    }
}
```

---

<sup>1</sup> One of the Java books in our library doesn't get to reading user input until page 316

Figure 6 Running public class Cost

```
bash-2.04$ java Cost
How much do they cost each3.95
How many do you want to buy1
Total cost 43.45 excluding tax and shipping
bash-2.04$
```



## Exercise

Write a program to prompt the user to enter a temperature in degrees Fahrenheit, and convert it to degrees Celsius.

To convert a temperature from Fahrenheit to Celsius

subtract 32

divide by 9

multiply by 5

Some sample results:

32    converts to 0

212   converts to 100

-40   remains -40

# *License*

*These notes are distributed under the **Well House Consultants Open Training Notes License**. Basically, if you distribute it and use it for free, we'll let you have it for free. If you charge for its distribution of use, we'll charge.*

### 3.1 Open Training Notes License

Training notes distributed under the **Well House Consultants Open Training Notes License** (WHCOTNL) may be reproduced for any purpose PROVIDE THAT:

- This License statement is retained, unaltered (save for additions to the change log) and complete.
- No charge is made for the distribution, nor for the use or application thereof. This means that you can use them to run training sessions or as support material for those sessions, but you cannot then make a charge for those training sessions.
- Alterations to the content of the document are clearly marked as being such, and a log of amendments is added below this notice.
- These notes are provided "as is" with no warranty of fitness for purpose. Whilst every attempt has been made to ensure their accuracy, no liability can be accepted for any errors of the consequences thereof.

Copyright is retained by Well House Consultants Ltd, of 404, The Spa, Melksham, Wiltshire, UK, SN12 6QL - phone number +44 (1) 1225 708225. Email contact - Graham Ellis (graham@wellho.net).

Please send any amendments and corrections to these notes to the Copyright holder - under the spirit of the Open Distribution license, we will incorporate suitable changes into future releases for the use of the community.

If you are charged for this material, or for presentation of a course (Other than by Well House Consultants) using this material, please let us know. It is a violation of the license under which this notes are distributed for such a charge to be made, except by the Copyright Holder.

If you would like Well House Consultants to use this material to present a training course for your organisation, or if you wish to attend a public course is one is available, please contact us or see our web site - <http://www.wellho.net> - for further details.

Change log  
Original Version, Well House Consultants, 2004

Updated by: \_\_\_\_\_ on \_\_\_\_\_  
Updated by: \_\_\_\_\_ on \_\_\_\_\_  
Updated by: \_\_\_\_\_ on \_\_\_\_\_  
Updated by: \_\_\_\_\_ on \_\_\_\_\_  
Updated by: \_\_\_\_\_ on \_\_\_\_\_  
Updated by: \_\_\_\_\_ on \_\_\_\_\_

*License Ends.*