

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

technical contact

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

Arrays

A regular variable can hold one piece of information, but an array can hold more than one. Arrays are declared to hold a number of elements, and data is then written to and read from each element based on its position number.

<i>Definition and declaration</i>	5
<i>Use</i>	5
<i>Array manipulation and replacement</i>	6
<i>Multidimensional arrays</i>	9
<i>Arrays of Objects</i>	14

There will be frequent occasions when you'll want to perform the same operation on a whole series of primitives or objects one after the other, and it won't be practical to hold each in a separate named variable.

Instead of using individual variables, we'll use a whole number of variables.

- We'll give the whole thing a single name
- We'll access elements by their numeric position in the table
- And we'll call the whole thing an **array**

Let's see an example. We'll:

- Create an array
- Read a number of float values into it
- Write out the values back-to-front

(Clearly one of those cases where we have to store what could be a lot of numbers.)

The program:

```
public class Arr1
{
    public static void main(String[] args)
    {
        int count = 0;
        float[] costs;
        costs = new float[5];

        while (count < 5)
        {
            System.out.print("Data: ");
            costs[count] = WellHouseInput.readNumber();
            if (costs[count] <= 0.0) break;
            count++;
        }

        for (int k=count-1; k>=0; k--)
            System.out.println(costs[k]);
    }
}
```

Figure 1 Running public class Arr1

```
bash-2.04$ java Arr1
Data: 4.3
Data: 2.4
Data: 4.5
Data: 0
4.5
2.4
4.3
bash-2.04$ java Arr1
Data: 12.7
Data: 100.4
Data: 30.4
Data: 2.8
Data: 2.7
2.7
2.8
30.4
100.4
12.7
bash-2.04$
```

Let's look through the various parts of that.

2.1 Definition and declaration

Just like other variables we have seen, you must define that the variable called "costs" is going to contain **float** information, and you must also declare it's going to be an array:

```
float[] costs;
```

That has defined how the variable name will be used, but has not set aside any memory for it.

Since computers store things one-after-another in memory, we must actually create our array, declaring its size:

```
costs = new float[5];
```

In Java, the array is actually an **object** (we'll come onto objects soon) created by using the **new** method.

It is important that you understand the distinction between defining an array and actually allocating the memory for it, although in practice you could do both in one line:

```
float [] costs = new float [5];
```

2.2 Use

We have defined that a variable name is to be used as an array, and we have set aside the memory for it. How do we actually make use of it?

Figure 2 Using an array

costs	
4.8	costs[0]
3.2	costs[1]
9.1	costs[2]
0	costs[3]
	costs[4]

We refer to the array name, and then the element number in square brackets. Elements are counted from 0 up, so our five-element array is numbered *zero* to *four*.

As well as referring to elements by giving an integer constant, we can use an integer variable, or even an integer expression within the square brackets, so that we calculate the element number.

And we can use array elements anywhere where we could use just a simple variable of the same type. Thus, in our sample program, we had:

```
costs[count] = WellHouseInput.readNumber();
```

which set an element of the array to contain the number read from the user, and:

```
if (costs[count] <= 0.0) break;
```

which used an element of the array in a calculation (in this case to yield a boolean

result), and:

```
System.out.println(costs[k]);
```

2.3 Array manipulation and replacement

You can choose to initialise the whole of an array and give an implicit size as follows:

```
/** Array initialization */

public class compton
{
    public static void main(String[] args)
    {
        int [] days =
            {31,28,31,30,31,30,31,31,30,31,30,31};
        System.out.print("Month number ... ");
        int month = (int) WellHouseInput.readNumber();
        if (month < 1 || month > 12)
        {
            System.out.println("Not a valid month number");
        } else {
            System.out.print("That month has ");
            System.out.print(days[month-1]);
            System.out.println(" days");
        }
    }
}
```

```
seal% java compton
Month number ...4
That month has 30 days
seal% java compton
Month number ...7
That month has 31 days
seal% java compton
Month number ...14
Not a valid month number
seal%
```

Figure 3 Running public class compton

Arrays are *objects* and the names you give them are *handles* which you use to reference those objects. A topic we'll see much more of during this course.

But for the moment, you might like to notice that

```
int[] numbers = days;
```

will give a second name to the array called "days".

Redefining an array will cause the old values and contents to be released and a fresh array object created.

Referring to numbers, **length** will tell you how many elements there are in an array.

```

/** Array manipulation */
public class upavon
{
    public static void main(String[] args) {
        int [] sizes =
            {31,28,31,30,31,30,31,31,30,31,30,31};
        System.out.print("number ... ");
        int month = (int) wellreader.read_number();
        if (month < 1 || month > 12)
            {
                System.out.println
                    ("Not a valid month number");
            } else {
                System.out.print("That month has ");
                System.out.print(sizes[month-1]);
                System.out.println(" days");
            }
        System.out.print("Length of sizes array: ");
        System.out.println(sizes.length);
        int day = month;
        sizes = new int[7];
        for (int k=0;k<7;k++) sizes[k]=24;
        if (day < 1 || day > 7)
            {
                System.out.println("Not a valid day number");
            } else {
                System.out.print("That day has ");
                System.out.print(sizes[day-1]);
                System.out.println(" hours");
            }
        System.out.print("Length of sizes array: ");
        System.out.println(sizes.length);
    }
}

```

Figure 4 Running public class upavon

```

seal% java upavon
number ... 3
That month has 31 days
Length of sizes array: 12
That day has 24 hours
Length of sizes array: 7
seal% java upavon
number ... 11
That month has 30 days
Length of sizes array: 12
Not a valid day number
Length of sizes array: 7
seal%

```

Exercise

Read in a series of up to 15 temperatures.

- Print them out in the order they were entered

Our example answer is flute

Sample

```
seal% java flute
Temperature, day 112.5
Temperature, day 216.4
Temperature, day 312.3
Temperature, day 421.5
Temperature, day 514.5
Temperature, day 613.4
Temperature, day 710.5
12.5 16.4 12.3 21.5 14.5 13.4 10.5
```

```
seal% java flute
Temperature, day 1-3.2
Temperature, day 21.2
Temperature, day 34.5
Temperature, day 4999
-3.2 1.2 4.5
```

- Print out the highest temperatures
- Print out the average of all temperatures

Our example answer is cornet

```
seal% java cornet
Temperature, day 140.5
Temperature, day 243.6
Temperature, day 350.3
Temperature, day 449.8
Temperature, day 5999
40.5 43.6 50.3 49.8
Highest is 50.3
Average is 46.05
```

For Advanced Students

- Take the first exercise and modify it to print out the numbers in ascending order

Hint: You'll do this by comparing pairs of numbers in the array and swapping them around until they're in order.

Our example answer is bass

```
seal% calc
Calculate what:(40.5+43.6+50.3+49.8)/4
46.05
seal% java bass
Temperature, day 140.5
Temperature, day 250.3
Temperature, day 344.1
Temperature, day 438.7
Temperature, day 5999
38.7 40.5 44.1 50.3
seal%
```

2.4 Multidimensional arrays

There will be times when you don't want just a row or column of numbers, but a whole table.

Easy ... just use two sets of square brackets!

Let's add this to what we've learnt so far and create, shuffle and deal a pack of 52 cards to 4 players:

```
// Well House Consultants2004.

import java.lang.Math;

/** Two Dimensional Array */

public class rushall
{
    public static void main(String[] args)
    {
        int [] pack = new int [52];
        int [] shpack = new int [52];
        int k,j;
        for (k=0;k<52;k++) pack[k]=k+1;

// shuffle

        for (k=0;k<52;k++)
        {
            int drawn_card = (int)
                (Math.random()*(52-k));
            shpack[k] = pack[drawn_card];
            pack[drawn_card]=pack[52-k-1];
        }
    }
}
```

Figure 5 Running public class rushall

```
seal% java rushall
Contents of hand number 1
 50 22 33 39 26 34 40 32 3 21 47 6 44
Contents of hand number 2
 42 28 11 8 14 51 12 23 18 37 35 38 52
Contents of hand number 3
 43 45 7 31 1 46 4 25 16 13 19 5 2
Contents of hand number 4
 41 10 36 17 24 29 48 9 15 49 27 20 30
seal% java rushall
Contents of hand number 1
 47 1 27 38 10 25 4 7 17 6 51 35 46
Contents of hand number 2
 49 37 50 26 3 24 33 14 29 19 30 36 23
Contents of hand number 3
 8 34 18 32 22 44 42 52 43 28 9 12 40
Contents of hand number 4
 31 2 15 39 11 41 16 13 21 45 5 20 48
seal%
```

The initial section creates a pack of 52 cards in order.

They are then shuffled by choosing a card at random 52 times over and adding it to the shuffled pack. As each card is added randomly to the shuffled pack, the last

remaining card in the unshuffled pack is moved up into its place.
Dealing is just distributing a card to each hand 13 times over ...

```
// deal

int [][] hands = new int [4][13];
int nc = 0;
for (j=0;j<13;j++) {
    for (k=0;k<4;k++) {
        hands[k][j]=shpack[nc++];
    }
}
```

Listing each hand is simply reading through the two-dimensional array and printing out the elements.

```
// list out hands

for (k=0;k<4;k++) {
    System.out.print("Contents of hand number ");
    System.out.println(k+1);
    for (j=0;j<13;j++) {
        System.out.print(" " + hands[k][j]);
    }
    System.out.println("");
}
}
```

Of course, there's lots more I could do.
What does this piece of code¹ do?

```
// Something else

for (k=0;k<4;k++) {
    for (j=12;j>=0;j--) {
        for (int i=0;i<j;i++) {
            if (hands[k][i]>hands[k][i+1]) {
                int holder = hands[k][i];
                hands[k][i]=hands[k][i+1];
                hands[k][i+1]=holder;
            }
        }
    }
}
```

If you need the code, it's *charlton.java*

Our card deck nicely divided into four hands of 13 cards - a rectangular array. What if we wanted to deal out three hands of five cards, then put the rest into the "bank", all in a two-dimensional array?

The first solution would be for us to set up our array size as 4 by 37 (52-15) and leave a whole load of elements unused.

The second solution is to allocate an array of arrays ... really what we have been doing anyway!

¹ We've added it before the hands are listed out.


```

seal% java wilsford
Contents of hand number 1
 38 51 39 52 3
Contents of hand number 2
 9 4 24 8 1
Contents of hand number 3
 5 29 26 4436
Contents of hand number 4
 22 42 11 17 6 35 46 20 19 16 48 10 21 2 18 23 40 30 27 25 34 43 32 7 41 12 15 14 47 50 49 28 37 31
seal%
    
```

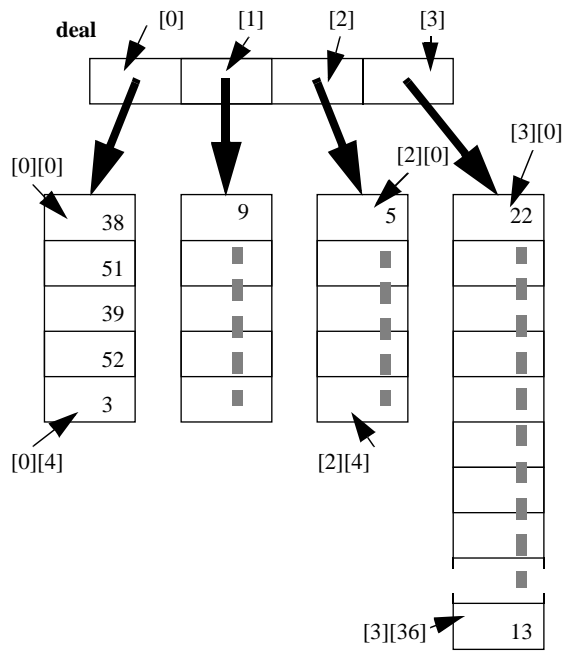


Figure 7 Putting data in via "wilsford" and how the deal array is structured

It's worth noting that, unlike static languages, we could decide the various array sizes as the program runs.

The following example is `Arr2`

```
// get number of players and size of hands
System.out.print("how many players? ");
int players = (int) WellHouseInput.readNumber();
System.out.print("how many cards each? ");
int cards = (int) WellHouseInput.readNumber();
```

And it'll make the actual code harder to read for newcomers.

```
// deal
int [][] deal = new int [players+1][];
int nc=0;

for (k=0;k<players;k++) deal[k] = new int [cards];
for (j=0;j<cards;j++) {
    for (k=0;k<players;k++) {
        deal[k][j]=shpack[nc++];
    }
}
deal[players] = new int [52-cards*players];
for (k=0;k<(52-cards*players);k++)
    deal[players][k] = shpack[nc++];
```

Figure 8 Running public class `Arr2`

```
bash-2.04$ java Arr2
how many players?4
how many cards each?7
Contents of hand number 1
 1 32 36 37 19 40 50
Contents of hand number 2
15 7 35 4 2 3 46
Contents of hand number 3
 5 22 18 25 6 31 41
Contents of hand number 4
43 12 51 34 42 27 39
Contents of hand number 5
11 33 45 21 23 8 49 20 10 26 16 30 48 24 44 28 52 13 47 14 38 9 2
bash-2.04$
```

This routine to list the hands out will now deal with any number of players, and any number of cards per hand:

```
// list out hands

for (k=0;k<players+1;k++) {
    System.out.print("Contents of hand number ");
    System.out.println(k+1);
    for (j=0;j<deal[k].length;j++) {
        System.out.print(" " + deal[k][j]);
    }
    System.out.println("");
}
}
```

We used the expression `players+1` to loop through each of the hands that we had defined. An alternative would have been to use `deal.length`.

For code that will be maintained by experienced programmers, `deal.length` would have been better code; it's much more maintainable and the same piece of code could be used to print out any two-dimensional array. However, if you're thinking of writing code like this, you need to be sure that future maintainers will understand the difference between

```
deal.length      (number of hands)
and
deal[k].length   (number of cards in hand k)
```

2.5 Arrays of Objects

Just as you can have arrays of primitives, so you can have arrays of objects too. The following tells you your longest and shortest films:

```
public class Weekend {

    public static void main(String [] args) {

        // Set up a series of film objects

        Film Watch[] = new Film[4];
        Watch[0] = new Film("Shrek",133);
        Watch[1] = new Film("Road to Perdition",117);
        Watch[2] = new Film("The Truth about Cats and Dogs",93);
        Watch[3] = new Film("Enigma",114);

        Film Longest = null, Shortest = null; // To avoid a grumpy compiler
        int longtime = 0, shorttime = 0; // To avoid a grumpy compiler

        for (int i=0; i<Watch.length; i++) {
            int mins = Watch[i].getminutes();
            if (i == 0) {
                Shortest = Longest = Watch[i];
                shorttime = longtime = mins;
            } else {
                if (mins < shorttime) {
                    shorttime=mins;
                    Shortest = Watch[i];
                }
                if (mins > longtime) {
                    longtime=mins;
                    Longest = Watch[i];
                }
            }
        }

        System.out.println("Longest film is " +
            Longest.getcalled() +
            " at "+longtime+" minutes");
        System.out.println("Shortest film is " +
            Shortest.getcalled() +
            " at "+shorttime+" minutes");
    }
}
```

Figure 9 Running public class
Weekend

```
bash-2.04$ java Weekend
Longest film is Shrek at 133 minutes
Shortest film is The Truth about Cats and Dogs at 93 minutes
bash-2.04$
```

Exercise

Read a series of temperatures for 06:00, 12:00, 18:00 and 00:00 for two days of the weekend.

Print the rise in temperature from 06:00 to 12:00 each day.

Our example answer is horn

```
seal% java horn
Temperature 1, day 112.5
Temperature 2, day 116.5
Temperature 3, day 114.6
Temperature 4, day 110.3
Temperature 1, day 213.6
Temperature 2, day 216.1
Temperature 3, day 215.3
Temperature 4, day 209.8
Temperature rise, day 1 was 4.0 degrees
Temperature rise, day 2 was 2.5 degrees
seal%
```

For Advanced Students

Create a table of binomial co-efficients (Pascal's triangle) in an appropriate array. The triangle starts as follows:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

Ask the user for a row and column number and print out the value.

Our example answer is hurdygurdy

```
seal% java hurdygurdy
Row (from 0)8
column (from 0)6
value is 28
seal% java hurdygurdy
Row (from 0)15
column (from 0)7
value is 6435
seal% java hurdygurdy
Row (from 0)3
column (from 0)3
value is 1
seal%
```

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.