An EasyPeasy Publication [Version 0.0.0.1]

C:\A Beginner's Guide to\Coding

C:\Learn How To\read code like a pro

C:\Learn How To\write code like a pro

C:\Learn How To\rub your temples and say 'this code
doesn't make any sense' like a pro

C:\ArrayList

An ArrayList is an adaptable array. It is a series of memory locations – or 'boxes' – each of which holds a single item of data, but with each box sharing the same name.

The easiest way to think of an ArrayList is to think of it as a table. For example, here we have set up a new ArrayList of Strings (text values) and called the ArrayList capitalCities. We have put three values into the array using the ArrayList.add() method (explained on the next page).

```
ArrayList<String> capitalCities = new ArrayList<String>()

capitalCities.add("London") capitalCities.add("Tokyo")
capitalCities.add("Ulaanbaatar")
```

So if we visualise this ArrayList as a table it looks like this:

capitalCities		
ArrayList Position 0	ArrayList Position 1	ArrayList Position 2
London	Tokyo	Ulaanbaatar

It is important to note that an ArrayList starts with an item in Position 0 and works up from there.

C:\ArrayList\add

We mentioned the ArrayList.add() method on the previous page.

ArrayLists are adaptable - you can add or remove elements from anywhere in the list at any time.

Here is a simple example of this using the established ArrayList capitalCities from the previous page:

```
capitalCities.add("Berlin")
```

capitalCities			
ArrayList Position 0	ArrayList Position 1	ArrayList Position 2	ArrayList Position 3
London	Tokyo	Ulaanbaatar	Berlin

If there are currently no entries into the ArrayList when you call the .add() method then it will add the first at position 0 and work up from there (as mentioned on the previous page).

C:\initialising variables

At the top of any program, the initial variables are defined.

A number is often defined as an int (integer):

E.g.

```
int number = 5
int secondNumber = 3
```

This sets up two variables, one called number which is equal to 5 and one called secondNumber which is equal to 3. Variable names can be anything you like.

These variables can be updated later on in the program.

E.g.

```
int number = 5

number minus 1 <-- once established you don't need to say int again
```

Now the integer number is equal to 4.

C:\while loops

A ‘while loop’ is a piece of code that continues to run while a certain argument remains valid.

While the statement in the brackets continues to be correct, then the code inside the { } continues to loop back on itself and runs again and again until the statement is no longer correct.

E.g.

```
int number = 5

while (number isSmallerThan 7)
{
   number plus 1
   System.out.print(“x”)
}
```

In this example, the integer number has been established as the number 5. When the code gets to the while loop it checks if the number is smaller than 7 and if it is then it runs the code in the {}.

Once the code in the brackets has run, it goes back to the top and checks again if number is smaller than 7. If it is, it runs again.

C:\while loops\continued

This is how the code on the previous page will run:

First time – number is 5, therefore smaller than 7, therefore the while loop runs, number is increased by 1 - from 5 to 6 and an “x” is printed out.

Second time - number is 6, therefore smaller than 7, therefore the while loop runs, number is increased by 1 - from 6 to 7 and a second “x” is printed out.

Third time – number is 7, which is no longer smaller than 7, therefore the while loop doesn’t run and the program moves on to whatever code is underneath the { } of the while loop.

Sometimes while loops can have two arguments –

E.g.

```
while (number isSmallerThanOrEqualTo 6 & number isBiggerThan 1)
```

– while both these statements are correct, the code in the { } runs.

C:\if statements

An ‘if statement’ is exactly what it sounds like.

If the statement in the () brackets after the word if is correct, then the code inside the { } runs.

If the statement is not correct then the code inside the { } doesn’t run and the program moves on.

E.g.

```
int number = 4

if (number isEqualTo “4”) <-- number DOES equal 4, so the code runs
{

 System.out.print(“This code ran”) <-- the code prints out “This code ran”

}
```

In this example, the integer number has been established as the number 4, so the code inside the { } runs and prints out “This code ran”.

C:\printing out

Occasionally the program you are running will need to print out text to the console.

This can be done by writing System.out.print(variable)

E.g.

```
System.out.print(“This code ran”)
```

The text “This code ran” will be printed out in the console.

C:\ArrayList\get

Here we look at the ArrayList.get() method.

This method will give you the ArrayList entry at whatever position you specify between the brackets, as a result.

Here is an example using the ArrayList capitalCities from the previous page:

```
int arrayPosition = 2

capitalCities.get(arrayPosition)
```

This would return the value “Ulaanbaatar” as “Ulaanbaatar” is in Position 2 in the ArrayList.

capitalCities			
ArrayList Position 0	ArrayList Position 1	ArrayList Position 2	ArrayList Position 3
London	Tokyo	Ulaanbaatar	Berlin

N.B. the get() method returns the value, it does not overwrite the arrayPosition value


An EasyPeasy Publication [Version 0.0.0.1]
(c) 1999 EasyPeasy Corporation. All rights reserved.