Looping – an intro

- Topics
  - Why is looping needed
  - Different types of looping
    - While
    - Do ... While
    - For

While/Do while

- The two more simple looping techniques are
  - While
  - Do ... While
- The difference between them is the time of when the associated test of the loop
  - While = BEFORE the loop itself
    - The loop may not get executed
  - Do ... While = AFTER the loop
    - The loop will be executed at least once

PRACTICAL USES OF LOOPS

- To wait for the user to select an option
  - Keep showing a menu until user has selected “Exit”
- To process a specific list of items
  - Letters within a word – by combining the charAt() function, we can traverse through the string
- Once we know of arrays, looping will have a lot more applications

While

Structure

while(TEST IS TRUE)

{

DO WHAT IS IN HERE

}

A repeating if loop

1. Test first
2. Then do what is inside the {}'s
3. Repeat until test becomes false

Do … While

Structure

do

{

} while (TEST IS TRUE)

A looping structure that will be executed at least once

1. Do what is inside {}'s
2. Then test the condition – if true, repeat!

Practical applications of do … while

If both WHILE and DO … WHILE do nearly the same time, when would you really need the DO.. WHILE?

If you know for definite that the code will be executed at least once, the DO … WHILE could be used

Game menu option

```
// Code here

int menuChoice = 0; // Variable to get menu option

 Scanner option = new Scanner(System.in); // Scanner object option instantiated


      // do while loop with condition to keep going while Exit is not selected

       do{

          System.out.println("=== Menu ===");

          System.out.println("Select option by typing in relevant number");

          System.out.println("1. Quick Start");

          // Code for option 1

          System.out.println("2. Instructions");

          // Code for option 2

          System.out.println("3. Start new game");

          // Code for option 3

          System.out.println("4. Exit");

          menuChoice = option.nextInt(); // grab the input and write it to variable menuChoice

       }while (menuChoice != 4); // End of Do … While
```

Using a variable as a counter

- Initialise the variable to a certain value BEFORE the While loop
  - int i = 0;
- Within the While loop increment or decrement it
  - i++; or i = i+1; (Simple increment of 1)
- Have the terminating condition to be i ==certain value
  - while(i<10)
- Loop will be broken once i gets to 10

Structure of loops

Each of the loops can be compared to an if statement in both

- Structure
  - All have ()'s enclosing the test, followed by { }'s
- Behavior
  - KEEP executing whatever is in the {}'s IF the test in the ()'s is true

Local Variables

- If a variable is DECLARED within one specific block of code, it is known as a local variable
  - If
  - While / Do ... While
  - For
  - Switch
- If a new variable is declared WITHIN that block, it CANNOT be used outside of that block
  - Used specifically within that BLOCK of code
  - Once outside, cannot be used

Local Variables II

- But what if we need the value anyways
- Copy the value of the variable to another variable that has more scope

```
public static void main(String [] args){
    int x = 0;
    int z = 0;// Scope of z is all in main
        if (x = =0) {
        int y = 8;
        z = y;
        }
    System.out.println(y); // This line will fail. Change y to z
} // End of main
```

Uses of local variables

- To do a specific task, for example
  - Add up the numbers fed in
  - Do a specific calculation
- Good Practise
  - Copy Global Variable to Local Variable
  - Do all associated calculations then using the local variable
  - Global Variable is not upset

Example

Simple while loop to add sum of #s from 0-9

```
int amount = 0;
while(amount < 10)
    {
        int temp = temp + amount;
        amount++;
        if (amount = = 9) // N.B. This if is inside the while loop
        {
            System.out.println(“The total is “ +temp)
        } // End of if
} // End of while
```

Nesting

Nesting is the coding concept where a decision/looping construct contains 1/more decision/looping constructs within it

For example

- A nested IF statement (An if within an if)
- An if within a for
- A switch within a do … while

These can be extended BUT the main point is the logic must be right

Nesting II

Now things, if they already haven't, should get a bit more complex

PUTTING LOGIC IN ORDER WITH CODE!

You can't just put in what you think will work and hope for the best – the compiler will not replace the thinking for you e.g. throwing in a distinct if followed by a looping construct and hoping compiler will understand

This will only come with practice but …

Nesting III

… there are some general things to remember if nesting

1. An IF or a SWITCH will only be done once
2. So if you need to repeat that action, typically you will surround the IF/SWITCH with a looping construct (not always, but most times)

Getting used to formulating algorithms is essential, a point which will be covered very shortly

For loop

While we will be covering the For loop later in week, good to introduce concept

For loop is perhaps the most common looping technque

Inbuilt counting mechanism

```
For (test)
{
Process to be looped
}
```

Loops and debugging

- Loops are one of the most common areas in need of debugging
  - High potential for losing track, especially if using nested loops
- Debugging is the process of getting code that compiles, but doesn't behave correctly, to behave correctly
- Debugging options
  - Breakpoints – Checkpoints within code
  - Watches – Checks values of variables

Distinct Debugging tools in Netbeans

As well as printing values of variables, we can use certain options within the Debug menu

- Watches
- Breakpoints
- Step-into
- Step over

Links

Links

1. https://www.youtube.com/watch?v=6djggrlkHY8