

Computer Science 1-2

Problem Set #4: While Loops

A **while** loop gives us a way to repeat sections of code as many times as we want (or until certain conditions are met) with one command. Much like the **if** statement, the **while** loop consists of two parts: a condition that we are testing and the curly braces where our code lives. Here's a fragment of an example of a program that uses a loop to print out all of the multiples of three that are less than 100.

```
int multiple = 0;
while (multiple < 100) {
    multiple += 3;
    System.out.println(multiple);
} // closing the curly brace controlled by the while
```

It is important to consider how the program will both enter and exit a loop. The **while** loop will check to see if the condition stated is true before going through it even once ... so, like an **if** statement, if your case isn't "true" to begin with, the code inside the loop won't run. Take a second to make sure you know what the output of the program will be. If you need to, type it up and run it to see what happens.

Typically we control the exit from a loop with either a **counter** or a **flag**. A counter is just what it sounds like; you create a variable that keeps track of how many times you are going through the loop. When a certain limit is reached you finish the loop...here is an example:

```
int counter = 0; //Typically we initialize counters to 0
while (counter < 10) {
    counter++; //This is a short way of adding 1 to a variable
}
System.out.println("Final value of counter is " + counter);
```

Notice a couple of things about our counter loop: first off we initialize our counter to 0 and the "counter++" line. This is a common thing we'll see in programming; while the rest of the world likes to start counting at 1, programmers like to start counting at zero. There's a reason for this and it has to do with how computer hardware works...0 actually represents all of the bits being off in a number (we'll chat about this more in class), so 0 really represents an "all-off" state. A good candidate for a starting point!

counter++ is another important part of Java we'll want to get comfortable with quickly. It is really just a quick way of writing counter = counter+1; We use these counter loops a lot (especially later when we work with **for** loops and arrays) so it will save you a lot of typing to start using it right away. There are a ton of shorthand commands in Java. Look around!

A flag is a way of exiting the loop that is based on user input or the results of a calculation (this is sometimes called a **sentinel**). For example, in this simple bank account program that adds up deposits...it keeps asking for deposits until the user enters -1 to indicate they are finished entering numbers:

```
import java.io.*;
import java.util.Scanner;

class num {

public static void main(String args[]) {

    int thisDeposit = 0;
    int totalDeposit = 0;

    Scanner in = new Scanner(System.in);

    while (thisDeposit != -1) {
        System.out.println("Enter Amount of Deposit or -1 to quit");
        thisDeposit = in.nextInt();
        if (thisDeposit != -1) //We only add the deposit if it is positive
            totalDeposit = totalDeposit + thisDeposit;
    }

    System.out.println("Thanks for banking");
    System.out.println("Your total deposit is: " + totalDeposit);
} //main
} //num
```

So, what you have seen above is a **while** loop. This tests the condition before entering the loop. Another loop is the **do-while** loop. It enters the loop and executes it and then tests the condition **afterwards** to decide whether or not to go and do the loop again.

Example:

```
i = 0;

do {

    System.out.println("i is " + i);

    i = i+1;

while (i<10);

??What is the highest value of i printed out?
```

Also, just like `if` statements, it is possible to have nested `while` loops (a loop within a loop)...but we'll get to that later. Now, the problems!

- 1) **Add it up!** – Write a program that lets the user input an integer and then adds up all the numbers up to (and including) that number. For instance, if the user types 5, the program would add up the numbers and print the screen:
 $1 + 2 + 3 + 4 + 5 = 15$.
- 2) **Grading Program-** Create a program that lets the teacher enter grades on a test and then calculates the class average. The teacher should be able to enter as many grades as they want and then enter -1 to stop. Note: The grades are typical: 90-100 is an A, 80 -89 is a B etc.
- 3) **Grading Program part 2-** Modify the program above to keep track of how many students received each letter grade on the exam and display the result along with the class average.
- 4) **Reciprocal-** Write a program to take in a number from a user, find its reciprocal and add it to a running sum. The program should repeat this procedure 10 times. However, if the user enters 0, the loop should ask the user if they want to stop adding numbers. Print the final sum at the end of the program (it is fine if it is a decimal number).
- 5) **LCM/GCD-** Write a program that takes in two numbers from the user and finds the Least Common Multiple and the Greatest Common Divisor of those numbers.
- 6) **Square Root!** Go to the following web site and discover how the ancient Babylonians (by the way, much of our western math comes from the Babylonians) to discover how to compute a square root. It will require a while-loop. Note: the test-driver (me) must be able to enter an integer number to tell the program how many iterations I want. This will be clearer once you read the web page on this. Here it is:
<https://blogs.sas.com/content/iml/2016/05/16/babylonian-square-roots.html>
Direct questions to me.
- 7) **Challenge-** Modify question #4 so that it prints out the sum as an actual fraction rather than a decimal. Make sure the fraction is properly reduced!

ALSO: All programs must now run with the following structure:

Your main program should be a **loop**. Each program above should be in its' own **method** as discussed in the last assignment. Your main loop should then look something like:

Ask the user to enter a number that represents the program they want to run (using the numbers above) or a -1. If they enter any of the numbers above you go to the method that executes the program. If they choose -1 then you terminate the loop.

ALSO: You need to write a method whose job is to get an integer entered. It stays in the method until the user enters a valid integer. How do you do this? Right now you have a **scanner** which can get an integer, but if the user types in a double your program crashes. That isn't acceptable. So, write a method which does the above and returns an integer.

Suppose in one of your programs you want to get an integer into a variable. We abstract the problem like this:

```
x = getMeanInt();
```

Now, all I have to do is write a method whose job is to input an **int** and not come back till it has one.

Here are the basic guts of it:

```
public static int getMeanInt() {  
  
    // This method will ask the user for an integer and return what the user types. This assumes you have a  
    // scanner called in declared at the start of the program.  
  
    int retval = 0;  
  
    boolean goodint = false;  
  
    String dummy = new String();  
  
    while (!goodint) {  
  
        System.out.println("Please enter an integer");  
  
        if (in.hasNextInt()) {  
  
            retval = in.nextInt();  
  
            goodint = true;  
  
        }  
  
        else {  
  
            dummy = in.nextLine();  
  
            System.out.println("You didn't enter an integer");  
  
        }  
  
    }  
  
    return retval;  
  
}
```

So, new stuff here is the `hasNextInt()`. That's another method on a scanner which looks to see if the user input an integer. It returns **true** if the user did. If the user didn't enter an integer, we have the scanner just

pull the rest of the line into a dummy string (which we don't care about) and then tell the user he/she screwed up (nicely!), and then repeat the process until a good integer comes in.

Now, the method **returns** to the point of call and gives back the value you are returning. Notice it doesn't return unless the user successfully gives an integer! This helps keep your program from crashing when the user messes up.