Loops

REPEATING BLOCKS OF CODE

So now you know how to write code where something either happens or doesn't happen, but what if you want to do something multiple times? What if you want to write a calculator that can handle doing more than one math problem before shutting down? Seems pretty doable….right? That's where loops come in. Loops are the first time we'll see code where it will actually run the same line of code multiple times over and over again….it goes in circles.

WHILE LOOPS

The most straightforward kind of loop is called a while loop. It looks and works a lot like an if statement, but instead of running *if* a statement is true it runs multiple times *while* a statement is true. So let's look at a simple if statement to start with:

```
int x = 1;

if (x < 5) {
    x++;
}
System.out.println(x);
```

So this code initializes x to 1, and since 1 is less than 5, it runs the block of code inside the if statement, which in this case increments x from 1 to 2. So this code will print out 2.

So let’s take a look at what happens when we make this a loop

```
int x = 1;
while (x < 5) {
    x++;
}
System.out.println(x);
```

So x still starts as 1, and when it gets to the while line, that line still runs just like the if statement did, so since 1 is less than 5, the code inside the loop will run, so x goes from 1 to 2. What changes here is that when it gets to the end, it loops back around to the top and runs it again. Since 2 is still less than 5, it runs again, so x becomes 3. This loop happens again over again for 3 and 4 until x ends up as 5. At that point the x < 5 line runs and becomes false. At this point, the loop ends and the code will output 5.

Just like an if statement, you can put any boolean statement inside a while loop, so you can also think of running a piece of code that looks like this

```
int x = 1;
while (x != 1) {
    x++;
}
System.out.println(x);
```

Notice that this loop starts out as false, so it will, in fact, never run, and this code prints 1.

Avoiding infinite loops

One common coding mistake is to write an infinite loop. This means that the loop will run forever and your code will never end. The most obvious form of infinite loop is one that looks like this:

```
while (true) {
     System.out.println(“Infinite loop”);
}
```

Since the condition will trivially always be true, it will continue printing this line over and over again.

You can also get an infinite loop by forgetting to change the variable in the condition

```
int x = 1;
while (x < 2) {
     System.out.println(“Infinite loop”);
}
```

Or if you change the variable but in the wrong direction:

```
int x = 1;
while (x > 0) {
     x++;
     System.out.println(“Infinite loop”);
}
```

You need to make sure that the conditional you have will eventually change from true to false based on the code in your loop.

Do-while loops

There is another variant on while loops called do-while loops. They work the same as while loops with one small difference: the block of code inside the loop will always run once before the condition is checked, even if the condition is false.

Do-while loops look like this:

```
int x = 1;

do {

     x++;

} while (x < 2);

System.out.println(x);
```

So in this case, x is 1 to start with, it then gets incremented to 2, then gets to the while condition, and then it will break from the loop and print 2.

FOR LOOPS

The other major type of loop is a for loop. And it looks a little bit different than things we've looked at before so let's take a look at the format before explaining how it works:

```
for (int i = 0; i < 5; i++) {

     System.out.println(i);

}
```

Let's take a look at the different parts of the loop

Setup

This statement runs once before the loop runs. In this loop the setup is `int i = 0;` This statement initializes a variable that will be used for this loop. This variable cannot be used outside the loop.

Test condition

This is the second statement in the first line, here it's `i < 5`. This is the comparison line for the loop. This functions just like the test condition in a while loop, the loop will end when this condition is reached. If your setup doesn't pass this test, the loop will never run.

Statement block

This is the next thing that runs and is the actual block of code inside the loop. This can consist of any code, here it is the println.

Update condition

After the loop has completed, the final part of the first line, the update condition runs, which changes the loop variable, often this either to increment or decrement the variable.

Tracing the code

So the way this code runs is:

1. i gets initialized to 0
2. 0 is less than 5 so the loop gets executed
3. 0 gets printed
4. i gets incremented to 1
5. 1 is less than 5 so the loop gets executed
6. 1 gets printed
7. i gets incremented to 2
8. 2 is less than 5 so the loop gets executed
9. 2 gets printed
10. i gets incremented to 3
11. 3 is less than 5 so the loop gets executed
12. 3 gets printed
13. i gets incremented to 4
14. 4 is less than 5 so the loop gets executed
15. 4 gets printed
16. i gets incremented to 5
17. 5 is not less than 5 so the loop exits

NESTING

Note that any block of code can go inside any other block of code. So an if statement can go inside a loop, a loop can go inside an if statement, and a loop can go inside of a loop. A loop within a loop is called a nested loop. Most often nested loops are on different variables.

CHOOSING TYPES OF LOOPS

For the most part you could use any type of loop to do any type of code that you need a loop for, but certain types of loops work better for certain types of situations.

for loops

For loops are best used in cases where you need to do something a very specific number of times. Think about doing something **for** every item in a list, or doing something 10 times.

while loops

While loops are best used in cases where you may not know exactly how many times you have to run it. Think along the lines of searching for something (**while** you haven't found it yet), or **while** you are still driving the robot.

do-while loops

Do-while loops are best used as retry loops. Think about a game where you want users to play it at least once and then give them the option to play again.

BREAK

Sometimes you want to stop a loop from looping even when a condition is true. Let's imagine a situation where you want to find the first number whose square is bigger than 10,000. So we know that the answer must be less than 10,000, so we'll write a while loop and create a variable for the answer, and then store something when we find something that times itself is bigger than 10,000

```
int x = 0;

int ans;

while (x < 10000) {

     if (x * x > 10000) {

          ans = x;

     }

     x++;

}
```

```
System.out.println(ans);
```

But there's a problem with this code. Even after we find the real answer (101), it is going to keep going through the numbers larger than that and end up printing out 10,000. And while the square of 10,000 is larger than 10,000, it isn't the first such number for which this is true.

What we want to do is to break out of the loop once we find the answer. So we can make our code look like this:

```
int x = 0;
while (x < 10000) {
     if (x * x > 10000) {
          break;
     }
     x++;
}
System.out.println(x);
```

The keyword break will break out of the loop. The rest of the code beneath it doesn't run and the rest of the times through the loop do not run.

TRY IT OUT!

Solve these problems using at least one loop.

1. Countdown! Take a user input number and countdown to 0.
2. Add onto your calculator to prompt the user if they would like to try again.
3. If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. Find the sum of all the multiples of 3 or 5 below 1000. [The answer is 233168. Since you need to use a loop to solve this it shouldn't change the problem any different to know the solution.]
4. For a given n (does not need to be input), output n rows like this (notice these are right-justified):

```
     *
    **
   ***
  ****
 *****
******
```