Loops

Introduction

In this article from my free Java 8 course, I will discuss the use of loops in Java. Loops allow the program to execute repetitive tasks or iterate over vast amounts of data quickly.

Background

In Example 1, we have a Person class, that counts the number of Person objects constructed.

```
public class Person {

     private static int personCounter;

     public Person() {
          personCounter = personCounter++;
     }

     public static int numberOfPersons() {
          return personCounter;
     }
}
```

Example 1

Now let’s say we wanted to create four people and then confirm that we actually created four person objects.

```
@Test
public void shouldReturnNumberOfPersonsInLoop() {
     Person person1 = new Person();
     Person person2 = new Person();
     Person person3 = new Person();
     Person person4 = new Person();
     assertEquals(4, Person.numberOfPersons());
}
```

Example 2

In Example 2 we created each of these objects on an individual line of code. After that we could finally see how many times we've called the Person constructor. There is however, an easier way to do this: by using loops. In the rest of this article I will provide an overview of three types of loops: the for loop, the while loop, and the do while loop.

for Loop

The first loop I will discuss is the "for loop". It is called a for loop because it tells the program "execute this loop FOR a number of times". Our for loop has three sections.

The first section assigns and defines a variable, such as "int i = 0". It uses this variable to iterate. You can use any variable name, but short names are common. The letter 'i' is often used because it stands for the word "index".

The second section defines how many times we want to execute the code inside the for loop. In Example 3, we used 'i < 4'. It is important to remember that it is commonplace in programming to start index values with 0, not 1. So since we want our code to iterate four times, we say 'i < 4' meaning that the code will execute when our 'i' value is 0, 1, 2, or 3.

The last section defines the incrementation of our variable. It can either increase the value or decrease it. In our example, we are increasing it by 1 using 'i++' which increases our 'i' value by 1. This occurs after the code block inside the for loop is finished. If we were to print out our 'i' values inside the code of our for loop, it would print '0,1,2,3', not 4. 'i' would increment to 4 after our code runs for the fourth time, but since that does not satisfy the condition in the middle section of our for loop, the code inside the loop stops executing.

```
public void shouldReturnNumberOfPersonsInLoop() {
     Person person1;

     for (int i = 0; i < 4; i++) {
          person1 = new Person();
     }
     assertEquals(4, Person.numberOfPersons());
}
```

Example 3

To restate the logic of the for loop, initially ‘i’ is assigned a value of 0. After we execute our code inside the loop, ‘i’ is now incremented by 1, so now it has a value of 1. The condition we wrote in the middle section stipulates that ‘i < 4’ for our code to execute. Since that is evaluated to be true, the code executes again, increments i by 1 and keeps repeating. The first time the condition evaluates to false is when ‘i’ has a value of 4. At this point the loop is exited and we will have executed our code 4 times.

You can also increment by numbers larger than 1. Say we incremented the previous example using “i = i + 2”, you would only create two person objects. This is because we are now only going through the loop twice, when i = 0 and i = 2.

One of the things you should watch out for when creating loops is the possibility of an infinite loop. For example if our condition in Example 3 is “i < 1” and you continually decrease the value of ‘i’ using “i--”, the loop will go on forever, create lots of objects, and eventually will probably crash your PC. Each of the three sections of the for loop is optional. In fact, technically, you could even leave all three sections blank and provide only two semicolons. But be careful, that way you would create an infinite loop!

while Loop

Our second loop is the while loop. A while loop allows the code to repeatedly execute WHILE a boolean condition is met. The basic syntax for a while loop is shown below.

```
while(condition) {
     //Code that executes if the condition is true
}
```

Example 4

The while loop is useful when you have a condition that is being dynamically calculated. We could create the person object inside a while loop, by dynamically changing the value of a variable ‘i’.

```
public void shouldReturnNumberOfPersonsInLoop() {
     Person person1;
     int i = 0;

     while (i < 4) {
          person1 = new Person();
          i++;
     }
     assertEquals(4, Person.numberOfPersons());
}
```
Example 5

In Example 5, we again create four person objects. Again we start off with ‘i’ valued at 0 and increment it by 1 while ‘i<4’. When that condition becomes false and ‘i’’s value is 4, the loop is exited.

do while Loop

The third type of loop we’ll look at in this article is the do while loop. A do while loop executes the code at least once and then repeatedly executes the block based on a boolean condition. It functions like a while loop after the first iteration, which happens automatically. In Example 6, we create the four Person objects using a do while loop.

```
@Test
public void shouldReturnNumberOfPersonsInLoop() {
     Person person1;
     int i = 0;
     do {
          person1 = new Person();
          i++;
    } while (i < 4);

     assertEquals(4, Person.numberOfPersons());
}
```
Example 6

There is also a fourth type of loop, the for each loop. However, this loop cannot be covered until we discuss another topic, arrays. This loop is discussed in my article about arrays.

The loop you should choose depends on the logic you need. All four of the loops are equivalent in terms of functionality, but each loop expresses the logic in a different style. You should always try to use the loop that is easiest for someone reading your code to understand and that makes the most sense in context.

Thanks for reading!