CMSC131

The **for** Loop

Variable Scope

Iteration

There are times when you know in advance exactly how many times you want to iterate.

There are times when you want to iterate until a certain condition or event occurs.

In the first case, you are probably better off using a **for** loop…

The FOR Loop

There will be three components within a FOR loop; initialization, a test condition, an update.

```
for (initialization-here; condition-here; update-here) {
          body-code-here;
}
```

NOTE: The term "update" is the term that some web pages and textbooks uses. This is sometimes referred to as the counting expression or the increment. The term "update" is the most generic so technically the most accurate. We will almost always have the update either increment or decrement a value that is initialized and tested in the condition.

An example **For** loop

```
public static void main(String[] args) {
    int sum = 0;

    for (int iter=1; iter<=100; iter++) {
      sum = sum + iter;
    }

    System.out.println("The sum from 1 to 100 is "+sum);
}

//NOTE: We will want to choose more meaningful
//      iterator names in real problems.
```

Programs will combine things we see…

```
public class SumToN {
  public static void main(String[] args) {
    int sum = 0;
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter a number: ");
    int n = sc.nextInt();

    for (int iter=1; iter<=n; iter++) {
      sum = sum + iter;
    }

    System.out.println("The sum from 1 to "+n+" is "+sum);
  }
}
```

What will this do?

```
int val = 0;
for (int iter=1; iter<207; iter=iter+3) {
    val = val + iter;
}

System.out.println("We ended at "+iter);
```

What will it do?

0% A. Print 205

0% B. Print 207

0% C. Print 208

0% D. Not Compile

```
int val = 0;
for (int iter=1; iter<207; iter=iter+3) {
   val = val + iter;
}

System.out.println("We ended at "+iter);
```

Response Counter

Fastest Responders

Seconds	Participant	Seconds	Participant

Scoping Rules

In Java, variables and constants defined within a block are local to their block of code and those identifiers can only be used within that scope.

Variables that are defined within a for loop exist only within the scope of that loop.

For example, the variable ***iter*** in the earlier example.

When a variable goes "out of scope" its memory should be thought of as being "free" again and in Java it is no longer valid for you to attempt to access it.

What will this do?

```
for (int iter=1; iter<207; iter=iter+3) {
   int val = 0;
   val = val + iter;
}

System.out.println("Total was" + val);
```

What will it do?

0% A. Print 7107

0% B. Print 205

0% C. Print something else

0% D. Not Compile

```
for (int iter=1; iter<207; iter=iter+3) {
   int val = 0;
   val = val + iter;
}

System.out.println("Total was" + val);
```

Response Counter

Counting down…

```
for (int secondsToLaunch=10; secondsToLaunch>0; secondsToLaunch--) {
      System.out.println(secondsToLaunch);
}
System.out.println("Launch!");
```

Will the value 0 be printed by the above code?

Note: *Good descriptive variable name, but bad for font size and PowerPoint…*

Where to start counting?

As we proceed, we'll see that sometimes loops might go from 1 to n

```
for (int iter=1; iter<=n; iter++)…
```

or might go from 0 to n-1

```
for (int iter=0; iter<n; iter++)…
```

This is sometimes due to personal style preferences and sometimes due to the way in which those values will be used within the body of the loop. We will see an example of this when we get to the topic of arrays.

Which feels more "natural" to you?

0% A. Starting at 0.

0% B. Starting at 1.

0% C. Either is fine.

Response Counter