For Loop

Java contains loop constructs other than the while loop. One of these constructs is the for loop. Here is the general syntax:

```
for (<init stmt>; <boolean expression>; <increment stmt>)
   stmt;
```

As mentioned with the while loop, usually, there is more than one statement in the body of a for loop, thus, we will usually have a block of statements:

```
for (<init stmt>; <boolean expression>; <increment stmt>) {
   stmt1;
   stmt2;
    ...
   stmtn;
}
stmtA;
```

Here is how the computer executes the for statement:

1) Execute the initial statement.
2) Evaluate the boolean expression.
3) If it's true, execute statements 1 through n.
4) Do the increment statement.
5) Then go back to step #2
6) If it's false, skip over the loop body and continue execution with stmt A.

Same Examples Using a For Loop (inside of main)

```
public class sum {

   public static void main(String[] args) {

      int val ;
      int sum = 0;

      for (val = 1; val < 100; val = val+2) {
         sum = sum + val;
      }
      System.out.println("1+3+5+...+99 = "+sum);
   }
}
```

```
public class tip {

   final public static double TIP_RATE = 0.15;
   final public static int MAX_PRICE = 100;

   public static void main(String[] args) {

      double tip_amt;

      // Print out all tips until the maximum meal value.
      for (int m_value=1; m_value <= MAX_PRICE; m_value++) {

          tip_amt = m_value*TIP_RATE;
          System.out.println("On a meal of $"+t_value+
                                ", you should tip $"+tip_amt);
      }
   }
}
```

How to choose between a for and while loop

First, it should be noted that anything you can do with a for loop, you can also do with a while loop. But, in certain situations, a for loop is more succinct and easier to read than a while loop that would do the same thing. In the example from the last page:

```
for (int val = 1; val < 100; val = val+2) {
    sum = sum + val;
}
```

from that first line, we can immediately see the structure of the loop (in particular how many times it's going to run.) All information that determines all the loop iterations are neatly placed in one place. In the corresponding while loop, you have to look at three different places to get that information.

Thus, based on this example, it can be seen that loops that run a particular number of iterations based on the value of a variable translate well into for loops. Especially if the task you are completing uses the index variable for some calculation.

Here is an example of such a situation:

Let's assume we are trying to print out a triangle to the screen that looks like this:

```
*
**
***
...
**********
```

Notice that when we print out ith line, we also want to print i stars on that line. (Assume that row is declared at the beginning of the function.) That gives us this skeleton of code:

```
for (row=1; row<=10; row++) {
  //print row stars on one line
}
```

Now, the question becomes, how do we print a line with row stars. Well, can't we just do a for loop? (Assume that col is also declared at the beginning of the function.)

```
for (col=1; col<=row; col++)
   printf("*");
```

So, we have the following:

```
for (row=1; row<=10; row++) {
   for (col=1; col<=row; col++)
      printf("*");
}
```

Is there a problem with this?

Since we never print out a newline character, all of our stars will print out in the same row. Instead, when we print a row, we need to finish by also printing a newline character. Here is the fix:

```
for (row=1; row<=10; row++) {
   for (row=1; col<=row; col++)  {
      printf("*");
   }
   printf("\n");
}
```

(Incidentally, note that the inner set of parentheses aren't necessary, but the outer ones are. Why?)