Repetition, Looping
CS101

Last time we looked at how to use if-then statements to control the flow of a program. In this section we will look at different ways to repeat blocks of statements. Such repetitions are called loops and are a powerful way to perform some task over and over again that would typically be too much work to do by hand. There are several ways to construct loops. We will examine the while and for loop constructs here.

While Loop

The while loop allows you to direct the computer to execute the statement in the body of the loop as long as the expression within the parentheses evaluates to true. The format for the while loop is:

```
while (boolean_expression) {
        statement1;
        …
        statement N;
}
```

As long as the Boolean expression evaluates to true, statements 1 through N will continue to be executed. Generally one of these statements will eventually make the Boolean expression become false, and the loop will exit. Here is an example that prints the numbers from 1 to 10:

```
int x=1;

while (x<=10) {
        System.out.println(x);
        x++;                    // Same as x=x+1
}
```

If we wanted to print out 1,000,000 numbers we could easily do so by changing the loop! Without the while loop, we would need 1,000,000 different print statements, certainly an unpleasant task for a programmer.

There are two types of while loops that we can construct. The first is a *count-based* loop, like the one we just used above. The loop continues, incrementing a counter each time, until the counter reaches some maximum number. The second is an event-based loop, where the loop continues indefinitely until some event happens that makes the loop stop. Here is an example of an event based loop:

```
ViewFrame vf = new ViewFrame("Loop Test");
int sum=0, x=0;
vf.setVisible(true);
while (x!=-9999) {
        x = vf.readInt("Enter an integer");
        if (x != -9999) {
                sum = sum + x;
        }
}
vf.println("The sum of the numbers you entered is " + sum);
```

This loop will input a number and add it to sum as long as the number entered is not –9999. Once –9999 is entered, the loop will exit and the sum will be printed. This is an event-based loop because the loop does not terminate until some event happens – in this case, the special value of –9999 is entered. This value is called a *sentinel* because it signals the end of input. Note that it becomes possible to enter the sentinel value as data, so we have to make sure we check for this if we don't want it to be added to the sum.

What is wrong with the following code? Hint: It results in what is called an infinite loop.

```
int x=1, y=1;

while (x<=10) {
        System.out.println(y);
        y++;
}
```

Exercise: Write a program that outputs all 99 stanzas of the "99 bottles of beer on the wall" song.

For example, the song will initially start as:

99 bottles of beer on the wall, 99 bottles of beer,
take one down, pass it around,
98 bottles of beer on the wall.

Write a loop so that we can output the entire song, starting from ninety-nine and counting down to zero.

Example: What is the output of this code?

```
        int j=0, k=0, x=5;

        while (j<x) {
                System.out.print("*");
                j++;
        }
        System.out.println();                   // Prints a newline
        j=0;
        while (j<x-2) {
                System.out.print("*");
                k=0;
                while (k<x-2) {
                        System.out.print(".");
                        k++;
                }
                System.out.println("*");
                j++;
        }
        j=0;
        while (j<x) {
                System.out.print("*");
                j++;
        }
        System.out.println();
}
```

This last example illustrates the concept of nested loops. It is possible, and often desirable, to insert one loop inside another loop. When this is done, it is called a nested loop.

Do-While Loop

It turns out that we can do all of the looping we need with the while loop. However, there are a number of other looping constructs make it easier to write certain kinds of loops than others. Consider the do-while loop, which has the following format:

```
do {
        statement1;
        statement2;
        …
        statement N;
} while (Boolean_condition);
```

The do-while loop executes all of the statements as long as the Boolean condition is true. How is this different from the while-do loop? In the do-while loop, the computer **always** executes the body of the loop at least once before it checks the Boolean condition. In the while-do loop, the Boolean condition is checked **first**. If it is false, then the loop's body is never executed.

For example, we could rewrite the do-while loop using an if-statement and a while loop:

```
if (Boolean_condition) {
        do {
                statement1;
                statement2;
                …
                statement N;
        } while (Boolean_condition);
}
```

This would be equivalent to the do-while loop.

As an example, let's convert the while loop we wrote to input numbers into a do-while loop. In the original example, we had to add an if-statement to check for the sentinel value inside the loop (because we don't want to add –9999 to the sum). This can be rewritten using a do-while loop without requiring the if-statement:

```
ViewFrame vf = new ViewFrame("Loop Test");
int sum=0, x=0;
vf.setVisible(true);
do {
        sum = sum + x;
        x = vf.readInt("Enter an integer");
} while (x!=-9999);
vf.println("The sum of the numbers you entered is " + sum);
```

Since the input comes at the end of the loop after the sum, we won't be adding in –9999. If the user types –9999, the loop will exit. Note that the first time through the loop, sum will get set to sum + x. However, by initializing x to zero, sum remains the same at 0, so if the first value typed is –9999, we will still get sum=0 in both cases.

Another place where a do-while loop is useful is to print menus and check for valid input:

```
int i;
ViewFrame vf = new ViewFrame();
vf.setVisible(true);
vf.println("Main Menu.  Enter 1 to perform task one, or 2 to perform task two.");
do {
        i = vf.readInt("Enter choice");
} while ((i!=1) && (i!=2));
```

This loop will continue as long as the user types in something that is neither '1' nor '2'.

Exercise: What would happen if the loop was:

```
do {
        i = vf.readInt("Enter choice");
} while ((i!=1) || (i!=2));
```

The For Loop

The for loop is a compact way to initialize variables, execute the body of a loop, and change the contents of variables. It consists of three expressions that are separated by semicolons and enclosed within parentheses:

```
for (expression1; expression2; expression3)
        statement;
```

Where, statement might include a { block } of statements:

```
for (expression1; expression2; expression3) {
        Statement1;
        Statement2;
        …
}
```

All of the expression statements are optional!

Expression1 is used to set initial values, and can set multiple values separated by a comma.

Expression2 is the condition for the loop to continue (while this is true).

Expression3 contains any operations you'd like to do at the end of each iteration. Separate different instructions with a comma.

The for loop can be written in the following equivalent while-loop format:

```
expression1;
while (expression2) {
        statement;
        expression3;
}
```

Here are some typical uses of for-loops:

```
int sum, i, value;
ViewFrame vf = new ViewFrame();
vf.setVisible(true);
for (i=0, sum=0; i<10; i++) {
        value = vf.readInt("Enter an integer");
        sum = sum + value;
}
```

This snippet loops from 0 to 9 and keeps a sum of values input by the user and saved in the variable sum.

We could do the same thing but count backwards from 10:

```
for (i=10, sum=0; i>0; I--) {
        value = vf.readInt("Enter an integer");
        sum = sum + value;
}
```

This loop ends when i=0.

Note that sometimes we can use a for loop to do work without any body at all!

```
for (i=2; i<=1000; i=i*2);
```

This snippet of code produces the first power of 2 larger than 1000. Note where the semicolon is placed to avoid any body at all. All the work is done in the loop heading.

Here is an example showing that the expressions in the loop header are optional. The following is equivalent:

```
i=10; sum=0;
for (;i>0;) {
        value = vf.readInt("Enter an integer");
        sum += value;
        i--;
}
```

This is equivalent to the previous example we did with the initialization and loop decrement all contained within the loop header.

Finally, consider the following:

```
for (;;) {
        System.out.println("hi");
}
```

This is equivalent to an infinite loop. We will print out "hi" forever until the user stops it by hitting control-c or closing the window.

Exercise: Write a program that inputs from the user how many numbers she would like to enter. The program should then input that many numbers and computes the average of the numbers.

Exercise: Write a program that finds and prints all of the prime numbers between 3 and 100. A prime number is a number such that one and itself are the only numbers that evenly divide it (e.g., 3,5,7,11,13,17, …)

Here is some pseudocode:

```
Loop from i=3 to 100
        Set flag = IsAPrime
        Loop from j=2 to i-1        (could we loop to a smaller number?)
                If i divided by j has no remainder
                        Then j evenly divides into i and i is not a prime number
                                Set flag = IsNotAPrime
        If flag still equals IsAPrime then
                Print i "is a prime number".
```