Lesson 12…..The *while* & *do-while* Loops

The *while* loop is basically the same as the *for*-loop except the **initializing** and **step** expressions are **not** part of the *while*-loop basic structure. In the following code we show the basic structure (skeleton) of the *while*-loop:

```
while( j <= 79 )
{
        … some code that we want repeated…
}
```

We notice in the above code that the only part similar to the *for*-loop is the **control expression** *j <= 79*. The **initializing** and **step** expressions are **absent**. As with the *for*-loop, the *while*-loop keeps repeating **as long as the control statement is *true***.

Summing numbers:

Now, let's actually do something with a *while*-loop. We will begin with a *for*-loop that sums the numbers from 3 to 79 and then perform this same task with a *while*-loop:

```
int sum = 0, j;
for (j = 3; j <= 79; j++)
{
        sum = sum + j;
}
System.out.println(sum);  //3157
```

An equivalent *while*-loop:

Here's a *while*-loop that does the same thing:

```
int sum = 0;
int j = 3;  //initializing expression…not part of loop.
while (j <= 79)  //control expression…fundamental part of loop
{
        sum = sum + j;
        j++;  //step expression…we have to remember to put this in.
              //It's not part of the basic "skeleton" of a while-loop.
}
System.out.println(sum);
```

The *do-while* loop:

A *do-while* loop is exactly the same as a *while*-loop except the control expression is at the **bottom** of the loop rather that at the **top** as it is with the *while*-loop. Following is the skeleton of a *do-while* loop:

```
do
{
            …some code that gets repeated…
}
while( j<= 79);
```

Note that *while* is not inside the braces. Also, notice the **semicolon**. It is a common mistake to leave it off.

We will now re-implement the *for*-loop above that sums from 3 to 79 as a *do-while* loop:

```
int sum = 0;
int j = 3;  //initializing expression
do
{
        sum = sum + j;
        j++;  //step expression
}while (j <= 79);   //control expression
System.out.println(sum);  //3157
```

What's the difference?

The main difference between the *while* loop and the *do-while* loop is **where** the test for staying in the loop is made (the control expression).

while-loop → test is at the **top** of the loop
do-while-loop → test is at the **bottom** of the loop

The *break* statement:

If *break* is encountered inside a loop, the loop terminates regardless of the status of the control statement. Code execution continues with the first line of code following the loop structure.

The *continue* statement:

If *continue* is encountered inside **any** loop (*for*, *while*, or *do-while*), all remaining code in the loop is skipped for this particular iteration; however, looping continues in accordance with the control expression.

This is illustrated with the following code:

```
int j = 0, boxer =11;
while(j <10)
{
        j++;
        if (j != 5)
        {
                continue;
        }
        boxer = boxer + j;
}
System.out.println(boxer);  //16
```

No braces:

If a *while* loop has no braces then it is understood that **only** the very next line of code (or structure such as another loop, *switch*, or *if* structure) is to be iterated (repeated). Consider the following code examples:

```
while(control expression)  …is equivalent to…  while( control expression)
pk = pk +2;                                    {
x = 97;                                            pk = pk +2;
                                               }
                                               x = 97;
```

Exercise for Lesson 12

1. Show the basic skeleton of a *while* loop.

2. Show the basic skeleton of a *do-while* loop.

3. Implement the following *for*-loop as a *while* loop.

```
int m;
for (m = 97; m <= 195; m++)
{
        k = k * k + 3 * m;
        p = p + m +1;
}
```

4. Implement the following *for* loop as a *do-while* loop.

```
for (int v = 2; v <= 195; v*=3)
{
        k = k * k + 3 * v;
        q = Math.sqrt(q + v +1);
}
```

5. What is the loop control expression in the code segment below?

```
while (!done)
{
        if (i < 1)
        {done = true;}
        i--;
}
```

6. What is the error in the code segment below?

```
do;
{
        if (i < 1)
        {done = true;}
        i--;
}while (!done);
```

7. How many times will the loop below iterate?

```
int j = 0;
while(j < 50)
{
        System.out.println("Hello World!");
}
```

8. How many times will the loop below iterate?

```
int j = 25;
while (j <= 100 | | j >= 25)
{
        System.out.println("Temp variable =" + j);
        j++;
}
```

9. Identify the error(s) in the code below:

```
j = 155
while (!done)
{
        if (j <= 25)
                done = true;
        j = j – 5;
};
```

10. What will be the output of the following code:

```
int i = 0, j = 0;
while(i <= 3)
{
        for(j = 0; j <=2; j++)
        {
                System.out.print(i + "," + j + "    ");
        }
        i++;
}
```

11. What command would you use if something unusual happens in one of your loops and you wish to exit prematurely (even before the control expression says you can)?

12. What loop structure would you use if you want to guarantee that a test condition of the control expression be tested **before** the block of code inside the loop could execute?

13. What is printed when the following code runs?

```
double m = 92.801;
int j = 0;
do
{
        j = j + 2;
        if (j > -100)
                continue;
        m+=3;
}while(j < 6);
System.out.println(m);
```

14. Write a program that will prompt the user to enter an integer. The program should square the number and then print the squared number. Repeat this process until 0 is entered as input. Use a do-while-loop to do this.

while & *do-while* loops… Contest Type Problems

1. Which of the following imitates the action of the *for*-loop to the right?

A. int j =0;
   while(j<100){ j++; …some code…}
B. int j=0;
   while(j<100){…some code… j++;}
C. int j=0;
   do{…some code… j++;}while(j<100);
D. Both B and C
E. Both A and B

```
for(int j=0; j<100; j++)
{
        … some code …
}
```

2. How many times does this loop iterate?

A. 0
B. 1
C. 2
D. Infinite number of times
E. Both A and B

```
int z = 19;
while(z < 20)
{
        if(z<100)
            continue;
        z++;
}
```

3. What is the output if the initial value of *k* and *p* are both 0?

A. 0
B. 3
C. 2
D. 1
E. None of these

```
do  {
        if(k= =1)
        {
                p+=3;
        }
        k++;
        p--;
}while(k<3);
System.out.println(p);
```

4. How many times does this loop iterate if the value of the *boolean b* is **not** known?

A. None
B. 2
C. Can't be determined
D. Infinite number of times
E. None of these

```
boolean p = true;
int sum=0;
while(p)
{
        sum+=5;
        if(b || !b)
                break;
}
```

5. What type of loop would you use if the condition for staying in the loop needs to be tested **before** the loop iterates?

A. *for*-loop
B. *while*-loop
C. *do-while* loop
D. All of these
E. Both A and B