THE JAVA WHILE STATEMENT

The while statement loops until its truth value is false. It is a pretest loop – it tests the truth value before entering the loop and if the truth value is already false at the start of the loop, it cycles zero times.

<table>
<thead>
<tr>
<th>Java While Statement Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>while ( truth value )</td>
</tr>
<tr>
<td>statement to repeat</td>
</tr>
<tr>
<td>statement below</td>
</tr>
<tr>
<td>while ( truth value )</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>statement(s) to repeat</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>statement below</td>
</tr>
</tbody>
</table>

If you have only one statement to repeat then the braces { } are optional.

<table>
<thead>
<tr>
<th>Java While Statement Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>What the computer executes:</td>
</tr>
<tr>
<td>What is the truth value?</td>
</tr>
<tr>
<td>Execute statement or statements to repeat</td>
</tr>
<tr>
<td>Proceed to execute statement below</td>
</tr>
</tbody>
</table>

As illustrated in the picture, the computer begins execution of the while statement by (1) evaluating the truth value, which can be either true or false. If true, the computer (2) executes the statement or statements to repeat after which it repeats the cycle. If the truth value is false, the computer (3) stops cycling and proceeds to the statement below.
Example
This code illustrates the semantics of the while statement.

```
1 System.out.print( "Enter a number: " );
2 x = scanner.nextDouble( );
3 c = 0;
4 while ( x > 1.0 )
5 {
6   x /= 2.0;
7   c++;
8   System.out.println( c + ": " + x );
9 }
10 System.out.println( "Done" );
```

For user input of 16, the loop cycles until x reaches 1.

```
Enter a number: 16
1: 8.0
2: 4.0
3: 2.0
4: 1.0
Done
```

For user input of 1.5, the loop completes the entire cycle and exits from line 4 even though line 6 sets x to 0.75.

```
Enter a number: 1.5
1: 0.75
Done
```

For user input of 0, the loop never cycles.

```
Enter a number: 0
Done
```
Line Breaks, Indention and the { } Delimiters

When it comes to interpreting the meaning of your `while` loop, the Java compiler totally ignores line breaks and indentation. It uses the `{ }` delimiters to determine what’s to be repeated; if omitted, it takes the one statement immediately following the truth value as what’s to be repeated.

**Examples**

Each of the following code fragments has the exact same behavior, outputting the line:

```
5 4 3 2 1 BLAST OFF!
```

```
int c = 5;
while ( c > 0 )
    System.out.print( c-- + " " );
System.out.println( "BLAST OFF!" );
```

```
int c = 5;
while ( c > 0 )
    System.out.print( c-- + " " );
    System.out.println( "BLAST OFF!" );
```

```
int c = 5;
while ( c > 0 )
    System.out.print( c-- + " " );
    System.out.println( "BLAST OFF!" );
```

```
int c = 5;
while ( c > 0 )
    System.out.print( c-- + " " );
}
System.out.println( "BLAST OFF!" );
```

```
int c = 5;
while ( c > 0 )
    System.out.print( c-- + " " );
}
System.out.println( "BLAST OFF!" );
```
```java
int c = 5;
while ( c > 0 )
{
    System.out.print( c + " " );
    c--;
}
System.out.println( "BLAST OFF!" );
```

### Beginner Errors on While Syntax
The biggest error Java novices make writing a `while` statement is to put an extraneous semicolon (`;`) after the truth value. This creates an `infinite loop` – a loop that never quits because its truth value is always true.

**Example**
In the code fragment below, the semicolon on line 2 creates an infinite loop. The statement to be repeated is the `null statement` – one which does nothing – whose end is marked with the semicolon. When executed, each cycle of the loop leaves `c` unchanged and so it is always greater than 0.

```java
1   int c = 5;
2   while ( c > 0 ) ;
3   {
4       System.out.print( c + " " );
5       c--;
6   }
7   System.out.println( "BLAST OFF!" );
```
The second biggest error is to omit or misplace the `{ } delimiters, which can also result in an infinite loop.

**Example**
In the code fragment below, the missing `{ } delimiters create an infinite loop. The statement to be repeated is line 3, which prints c but doesn’t decrement it. When executed, each cycle of the loop leaves c unchanged and prints 5 repeatedly.

```
1   int c = 5;
2   while ( c > 0 )
3       System.out.print( c + " " );
4       c--;
5   System.out.println( "BLAST OFF!" );
```

Misplaced `{ } delimiters is a common mistake among novice programmers. You can help by making an effort to keep your code neatly and consistently indented.

**Exercises**

Take the code given in the first example of this topic (page 2) and fashion it into a complete working Java application. Enter it into jGRASP and save it to a file. Compile it and fix any syntax errors. Perform the following series of experiments and answer any questions.

1. In jGRASP’s main window, set a breakpoint at the line shown below. To set the breakpoint, move the mouse cursor to the gray bar at the left of your code. It will display a red dot ○. Click the mouse to set the breakpoint.

   ```java
   while ( x > 1.0 )
   ```

2. Start jGRASP’s debugger by clicking the Ladybug button 🐞. Execution will pause at the line shown below to allow you to enter input. Enter the input 8.

   ```java
   x = scanner.nextDouble();
   ```
3. After entering the input, program execution halts at the line:

```java
while ( x > 1.0 )
```

Notice that program execution halts immediately before the statement has been executed.

In jGRASP’s Debug pane, what are the values of variables \( x \) and \( c \)?

4. Resume execution by clicking jGRASP’s Resume button ➡️.

Notice that program execution again halts at the line:

```java
while ( x > 1.0 )
```

In jGRASP’s Debug pane, what are the values of variables \( x \) and \( c \)? What happened when you clicked the Resume button?

5. Repeatedly click the Resume button until the program prints **Done** and halts. What are the values of \( x \) and \( c \) after each click?

6. How many times did the loop cycle?

7. What is the output of the application?

8. Replay the experiment of problems 1 through 5, this time entering input of 1.5. How many times did the loop cycle? What is the output of the application?

9. Replay the experiment of problems 1 through 5, this time entering input of 0.5. How many times did the loop cycle? What is the output of the application?
### Question 10

What is the output of this code segment if the user enters 48? If the user enters 49? If the user enters 50?

```java
Scanner input = new Scanner( System.in );
int c = input.nextInt( );
while ( c < 50 )
{
    System.out.println( "YES" );
    System.out.println( "NO" );
    c++;
}
System.out.println( "DONE" );
```

### Question 11

For each `while` statement below, circle what’s wrong and explain. None of them is correct.

11. ```java
    Scanner input = new Scanner( System.in );
    int c = input.nextInt( );
    While ( c < 50 ) {
        System.out.println( "YES" );
        System.out.println( "NO" );
        c++;
    }
    System.out.println( "DONE" );
```

12. ```java
    Scanner input = new Scanner( System.in );
    int c = input.nextInt( );
    while c < 50 {
        System.out.println( "YES" );
        System.out.println( "NO" );
        c++;
    }
    System.out.println( "DONE" );
```
For each **while** statement below, circle what’s wrong and explain. None of them is correct.

<table>
<thead>
<tr>
<th></th>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| 13. | `Scanner input = new Scanner( System.in ); int c = input.nextInt( ); while ( c < 50 )
         System.out.println( "YES" );
         System.out.println( "NO" );
        c++;
        System.out.println( "DONE" ); | The statement `System.out.println( "YES" );
            System.out.println( "NO" );` should be inside the `while` loop. |
| 14. | `Scanner input = new Scanner( System.in ); int c = input.nextInt( );
          while ( c < 50 ) {
            System.out.println( "YES" );
            System.out.println( "NO" );
          }
          System.out.println( "DONE" ); | The closing curly brace `{` is missing. |
| 15. | `Scanner input = new Scanner( System.in ); int c = input.nextInt( );
          while ( c < 50 ) {
            System.out.println( "YES" );
            System.out.println( "NO" );
          } c++;
          System.out.println( "DONE" ); | The statement `c++;` should be outside the `while` loop. |
16. What is the output of this code segment if the user enters 50 20? If the user enters 50 50? If the user enters 20 50?

```java
Scanner input = new Scanner( System.in );
double x = input.nextDouble( );
double y = input.nextDouble( );
int r = 0;
while ( x >= y )
{
    x -= y;
    r++;
}
System.out.println( "x = " + x );
System.out.println( "y = " + y );
System.out.println( "r = " + r );
```

For each of the following exercises, write a Java `while` statement that does the job.

17. Print the even integers from 0 to 100.

18. Print the integers from 5 to 100 by 5s (e.g. 5, 10, 15, and so on).

19. Reads one word after another from `System.in` until the user enters the word `quit`. Hint: look up the description of method `equals` in the API specification of class `String`.

20. Repeat the previous problem, allowing the user to enter the word `quit` in any letter case. For example, `Quit`, `QUIT`, `quit` and `QuIt` all cause the loop to stop. Hint: look up the description of method `equalsIgnoreCase` in the API specification of class `String`. 

The Java While Statement