THE JAVA DO-WHILE STATEMENT

The do-while statement loops until its truth value is false. It is a posttest loop – it tests the truth value after the first loop cycle. Therefore, it always cycles at least once.

Java Do-While Statement Syntax

<table>
<thead>
<tr>
<th>do</th>
<th>do</th>
</tr>
</thead>
<tbody>
<tr>
<td>do</td>
<td>do</td>
</tr>
<tr>
<td>statement to repeat</td>
<td>statement(s) to repeat</td>
</tr>
<tr>
<td>while ( truth value );</td>
<td>while ( truth value );</td>
</tr>
<tr>
<td>statement below</td>
<td>statement below</td>
</tr>
</tbody>
</table>

If you have only one statement to repeat then the braces {} are optional. The entire statement must end with a semicolon ;.

Java Do-While Statement Semantics

What the computer executes:

2 true  
Execute statement or statements to repeat

What is the truth value?

3 false  
Proceed to execute statement below

As illustrated in the picture, the computer begins execution of the do-while statement by (1) executing the statement or statement(s) to repeat after which it evaluates the truth value, which can be either true or false. If true, the computer (2) 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 do-while statement.

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

For user input of 16, the loop cycles until \( x \) reaches 1.
For user input of 1.5, the loop completes the entire cycle and exits from line 10 even though line 6 sets \( x \) to 0.75.
For user input of 0, the loop still cycles once.

<table>
<thead>
<tr>
<th>Enter a number: 16</th>
<th>Enter a number: 1.5</th>
<th>Enter a number: 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: 8.0</td>
<td>1: 0.75</td>
<td>1: 0.0</td>
</tr>
<tr>
<td>2: 4.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3: 2.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Done</td>
<td>Done</td>
<td>Done</td>
</tr>
</tbody>
</table>
**Line Breaks, Indentation and the { } Delimiters**
The Java compiler ignores line breaks and indentation as it interprets the meaning of your `do-while` loop. The `{ }` delimiters are only necessary if you have more than one statement to repeat.

**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;
do
    System.out.print( c-- + " " );
while ( c > 0 );
System.out.println( "BLAST OFF!" );
```

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

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

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

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

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

int c = 5;
do {
    System.out.print( c + " " );
    c--;
} while ( c > 0 );
System.out.println( "BLAST OFF!" );
The Do-While Statement is Novice Friendly
The **do-while** statement is good for novice programmers because the compiler complains about stray semicolons and missing `{ }` delimiters.

**Example**
For the code fragment below, in which the `{ }` delimiters are incorrectly omitted, the compiler issues the diagnostic:

```
MyApp.java:3: while expected
    System.out.print( c + " " );
       ^
```

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

**Example**
For the code fragment below, which has a stray semicolon on line 2, the compiler issues the diagnostic:

```
MyApp.java:2: while expected
    do ;
       ^
```

```
1 int c = 5;
2 do ;
3 {
4   System.out.print( c + " " );
5   c--;
6 }
7 while ( c > 0 );.out.println( "BLAST OFF!" );
```
## 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? What happened after you entered input?

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 \texttt{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 \texttt{0.5}. How many times did the loop cycle? What is the output of the application?

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

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

For each \texttt{do-while} statement below, circle what’s wrong and explain. None of them is correct.

11. \texttt{Scanner input = new Scanner( System.in );}
    \texttt{int c = input.nextInt( );}
    \texttt{Do}
    \texttt{[b\{ \}}
    \texttt{System.out.println( "YES" );}
    \texttt{System.out.println( "NO" );}
    \texttt{c++;}
    \texttt{\}}
    \texttt{While ( c < 50 );}
    \texttt{System.out.println( "DONE" );}
For each **do-while** statement below, circle what’s wrong and explain. None of them is correct.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 12. | `Scanner input = new Scanner( System.in );
    int c = input.nextInt( );
    do
    {
        System.out.println( "YES" );
        System.out.println( "NO" );
        c++;
    }
    while c < 50;
    System.out.println( "DONE" ); |
| 13. | `Scanner input = new Scanner( System.in );
    int c = input.nextInt( );
    do
    {
        System.out.println( "YES" );
        System.out.println( "NO" );
        c++;
    }
    while ( c < 50 )
    System.out.println( "DONE" ); |
| 14. | `Scanner input = new Scanner( System.in );
    int c = input.nextInt( );
    do
    System.out.println( "YES" );
    System.out.println( "NO" );
    c++;
    while ( c < 50 );
    System.out.println( "DONE" ); |
For each `do-while` statement below, circle what’s wrong and explain. None of them is correct.

| 15. | `Scanner input = new Scanner( System.in );`  
|     | `int c = input.nextInt( );`  
|     | `do`  
|     | `{`  
|     | `System.out.println( "YES" );`  
|     | `System.out.println( "NO" );`  
|     | `}`  
|     | `while ( c < 50 );`  
|     | `System.out.println( "DONE" );` |

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;
do  
{  
x -= y;
    r++;
} while ( x >= y );
System.out.println( "x = " + x );
System.out.println( "y = " + y );
System.out.println( "r = " + r );
```