## THE JAVA FOR STATEMENT

The **for** statement behaves as a **while** statement. Its syntax visually emphasizes the code that initializes and updates the loop variables.

### Java For Statement Syntax

```
for (init; truth value; update)  
statement to repeat  
statement below
```

```
for (init; truth value; update)  
{  
statement(s) to repeat  
}  
statement below
```

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

### Java For Statement Semantics

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Execute <code>init</code></td>
</tr>
<tr>
<td>2</td>
<td>What is the <code>truth value</code>?</td>
</tr>
<tr>
<td>3</td>
<td>If <code>true</code>, execute <code>statement or statements to repeat</code></td>
</tr>
<tr>
<td>4</td>
<td>Execute <code>update</code></td>
</tr>
<tr>
<td>5</td>
<td>If <code>false</code>, execute <code>update</code></td>
</tr>
<tr>
<td>6</td>
<td>Proceed to execute <code>statement below</code></td>
</tr>
</tbody>
</table>

As illustrated in the picture, the computer begins execution of the **for** statement by (1) executing the `init`. (2) The computer evaluates the `truth value`, which can be either `true` or `false`. If `true`, the computer (3) executes the `statement or statements to repeat`, (4) executes the update and (5) repeats the cycle. If the `truth value` is `false`, the computer (6) stops cycling and proceeds to the `statement below`. 
**Example**
This code illustrates the semantics of the \texttt{for} statement.

```
System.out.print( "Enter a number: " );
last = scanner.nextInt( );
for ( int c = 1; c <= last; c++ )
{
    System.out.println( c + " Mississippi" );
}
System.out.println( "Done" );
```

For user input of 5, the loop cycles for \( c = 1, 2, 3, 4 \) and 5.
For user input of 1, the loop cycles once.
For user input of 0, the loop never cycles.

<table>
<thead>
<tr>
<th>Enter a number: 5</th>
<th>Enter a number: 1</th>
<th>Enter a number: 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Mississippi</td>
<td>1 Mississippi</td>
<td>Done</td>
</tr>
<tr>
<td>2 Mississippi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Mississippi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Mississippi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 Mississippi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Done</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The \texttt{for} statement is particularly useful for constructing a \textit{definite counting loop}, which is a loop that cycles a definite number of times. Such a loop uses a variable called the \textit{loop control variable} to keep count of how many cycles it’s been through. Here’s its general form written using a \texttt{for} statement.

**Form of a Definite Counting Loop Using a for Statement**

```
for ( \texttt{lcv} = 1; \texttt{lcv} <= \texttt{count}; \texttt{lcv} ++ )
{
    \textit{statement(s) to repeat}
}
```

\( \texttt{lcv} \) is the loop control variable.
\( \texttt{count} \) is the desired number of cycles.
Line Breaks, Indentation and the { } Delimiters
When it comes to interpreting the meaning of your for loop, the Java compiler totally ignores line breaks and indention. 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 lines:

```
 1 potato
2 potato
3 potato
FOUR
```

```
for ( int k = 1; k < 4; k++ )
   System.out.println( k + " potato" );
System.out.println( "FOUR" );
```

```
for ( int k = 1; k < 4; k++ )
System.out.println( k + " potato" );
System.out.println( "FOUR" );
```

```
for ( int k = 1; k < 4; k++ )
   System.out.println( k + " potato" );
   System.out.println( "FOUR" );
```

```
for ( int k = 1; k < 4; k++ )
   System.out.println( k + " potato" );
   System.out.println( "FOUR" );
```

```
for ( int k = 1; k < 4; k++ )
   System.out.println( k + " potato" );
   System.out.println( "FOUR" );
```

```
for ( int k = 1; k < 4; k++ )
   System.out.println( k + " potato" );
}
System.out.println( "FOUR" );
```

```
for ( int k = 1; k < 4; k++ )
{
   System.out.println( k + " potato" );
}
System.out.println( "FOUR" );
```
for ( int k = 1; k < 4; k++ )
{
    System.out.print( k );
    System.out.println( " potato" );
}
System.out.println( "FOUR" );

for ( int k = 1; k < 4; k++ ) {
    System.out.print( k );
    System.out.println( " potato" );
}
System.out.println( "FOUR" );
**Accessing the For Loop Control Variable**

If you declare the `for` statement’s loop control variable above the loop then you can check its value after the loop quits.

---

**Example**

The code fragment below prints the value of `k` at line 5. `k` is declared at line 1. The output is:

```
1 potato
2 potato
3 potato
FOUR
4 potatoes!
```

```
1 int k;
2 for ( k = 1; k < 4; k++ )
3     System.out.println( k + " potato" );
4     System.out.println( "FOUR" );
5     System.out.println( k + " potatoes!" );
```

However, if you declare the loop control variable within the `init` clause then it is local to the `for` statement and cannot be accessed outside of it.

---

**Example**

In the code fragment below, `k` is declared in the `init` clause of the `for` statement (see line 1) and is, therefore, unknown outside it. The compiler issues this diagnostic for line 4:

```
MyApp.java:4: cannot find symbol
  symbol : variable k
location: class MyApp
  System.out.println( k + " potatoes!" );
  ^
```

```
1 int k;
2 for ( int k = 1; k < 4; k++ )
3     System.out.println( k + " potato" );
4 System.out.println( "FOUR" );
5 System.out.println( k + " potatoes!" );
```
Beginner Errors on For Syntax
The biggest error Java novices make writing a `for` statement is to put an extraneous semicolon (;) at the end of the line. This usually causes a syntax error because, by ending the `for` statement, any reference to the counting variable is out of scope.

Example
Because of the semicolon at the end of line 1, the compiler believes that line 2 is not part of the `for` statement. Thus, at line 2, `k` is unknown. The compiler issues this diagnostic:

```
MyApp.java:2: cannot find symbol
  symbol   : variable k
  location: class MyApp
    System.out.println( k + " potato" );
    ^
```

```
1  for ( int k = 1; k < 4; k++ ) ;
2       System.out.println( k + " potato" );
3       System.out.println( "FOUR" );
```

The second biggest error is to omit or misplace the `{ }` delimiters.

Example
The code fragment below only repeats line 2, thus printing:

```
123 potato
FOUR
```

```
1  for ( int k = 1; k < 4; k++ )
2       System.out.print( k );
3       System.out.println( " potato" );
4       System.out.println( "FOUR" );
```

Misplaced `{ }` delimiters is a common mistake among novice programmers. You can help by making an effort to keep your code neatly and consistently indented.
Multiple Operations in the For Statement Clauses
You can place multiple operations, separated by commas, in the *init* and *update* clauses of the *for* statement.

**Example**
The code fragment below prints:

1 + 12 = 13
2 + 11 = 13
3 + 10 = 13
4 + 9 = 13
5 + 8 = 13
6 + 7 = 13

```java
int i, j;
for ( i=1, j=12; i <= j ; i++, j-- )
   System.out.println( i + " + " + j + " = " + (i+j) );
```

**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
   for ( int c = 1; c <= last; c++ )
   ```

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 4.

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

   ```java
define ( int c = 1; c <= last; c++ )
```

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

In jGRASP’s Debug pane, what is the value of variable last? Why is variable c not shown in the Debug pane?

4. Resume execution by clicking jGRASP’s Resume button [Resume].

Notice that program execution again halts at the line:

   ```java
define ( int c = 1; c <= last; c++ )
```

In jGRASP’s Debug pane, what is the value of variable c? What happened when you clicked the Resume button?

5. Repeatedly click the Resume button until the program prints Done and halts. What is the value of 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. 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. 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 2? If the user enters 1? If the user enters 0?

```java
Scanner input = new Scanner( System.in );
int last = input.nextInt( );
for ( int k = 1; k <= last; k++ )
{
    System.out.println( "YES" );
    System.out.println( "NO" );
}
System.out.println( "DONE" );
```

11. What is the output of this code segment if the user enters 2? If the user enters 1? If the user enters 0?

```java
Scanner input = new Scanner( System.in );
int last = input.nextInt( );
for ( int c = last; c > 0; c-- )
    System.out.print( c + " " );
System.out.println( "BLAST OFF!" );
```

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

12. ```java
Scanner input = new Scanner( System.in );
int last = input.nextInt( );
For ( int c = last; c > 0; c-- )
    System.out.print( c + " " );
System.out.println( "BLAST OFF!" );
```

13. ```java
Scanner input = new Scanner( System.in );
int last = input.nextInt( );
for ( int c = last, c > 0, c-- )
    System.out.print( c + " " );
System.out.println( "BLAST OFF!" );
```
For each `for` statement below, circle what’s wrong and explain. None of them is correct.

<table>
<thead>
<tr>
<th>Number</th>
<th>Code Segment</th>
</tr>
</thead>
</table>
| 14. | ```java
Scanner input = new Scanner( System.in );
int last = input.nextInt( );
for ( c = last; c > 0; c-- )
    System.out.print( c + " " );
System.out.println( "BLAST OFF!" );
``` |
| 15. | ```java
Scanner input = new Scanner( System.in );
int last = input.nextInt( );
for ( int c = last; c > 0; c-- ) ;
    System.out.print( c + " " );
System.out.println( "BLAST OFF!" );
``` |

What is the output of each of these code segments?

<table>
<thead>
<tr>
<th>Number</th>
<th>Code Segment</th>
</tr>
</thead>
</table>
| 16. | ```java
System.out.println( "Three cheers for Ironman!" );
for ( int i = 1; i <= 3; i++ )
    System.out.println( "HIP HIP, HOORAY!" );
``` |
| 17. | ```java
System.out.println( "Three cheers for Ironman!" );
for ( int i = 1; i <= 3; i++ ) ;
    System.out.println( "HIP HIP, HOORAY!" );
``` |
| 18. | What is the output of this code segment if the user enters 10 3? If the user enters 3 3? If the user enters 3 10? ```java
Scanner input = new Scanner( System.in );
int f = input.nextInt( );
int s = input.nextInt( );
int r = 0;
for ( int k = s ; k <= f ; k += s )
    r = (r * 10) + k;
System.out.println( "r = " + r );
``` |
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?

declare x, y;  
declare int r;  
Scanner input = new Scanner( System.in );  
x = input.nextDouble();  
y = input.nextDouble();  
for ( r = 0; x >= y; x -= y, r++) ;  
System.out.println( "x = " + x);  
System.out.println( "y = " + y);  
System.out.println( "r = " + r);