NESTED CONTROL STATEMENTS

A *nested control statement* is a control statement that is contained within another control statement. You can do this to many levels, building up elaborate composites of various control statements. The compiler interprets these hierarchically, treating a composite control statement as a single statement for the purpose of determining what is affected by the next higher level of control.

**Example**
Consider this Java fragment containing a *for* statement nested inside another.

```java
for ( int i=0; i<3; i++ )
    for ( int j=0; j<4; j++ )
        System.out.println( i + " " + j );
```

To understand how the compiler parses this construct, let’s review the syntax of the *for* statement:

<table>
<thead>
<tr>
<th>Java For Statement Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>for ( init; truth value; update )</code></td>
</tr>
<tr>
<td><code>statement to repeat</code></td>
</tr>
<tr>
<td><code>statement below</code></td>
</tr>
</tbody>
</table>

Layering this format onto the code fragment, we see that the compiler parses it as shown in the picture below, treating the inner *for* statement as a single statement being repeated by the outer *for* statement.
Thus, the `println` method is executed a total of 12 times: 4 times ($j = 0, 1, 2$ and 3) by the inner `for` statement, which itself is executed 3 times ($i = 0, 1$, and 2) by the outer `for` statement. The output is:

```
0 0
0 1
0 2
0 3
1 0
1 1
1 2
1 3
2 0
2 1
2 2
2 3
```

**Example**

This Java fragment contains an `if-else` statement nested inside a `for` statement.

```java
int letterCnt = 0;
int nonLetterCnt = 0;
for (int c=0; c<input.length(); c++)
    if (Character.isLetter(input.charAt(c)))
        letterCnt++;
    else
        nonLetterCnt++;
```

To understand how the compiler parses this construct, let’s review the syntax of the `if-else` statement. The `for` statement syntax is shown in the previous example.

<table>
<thead>
<tr>
<th>Java If-Else Statement Syntax</th>
</tr>
</thead>
</table>
| `if ( truth value )
true path
else
false path
statement below` |
Layering these syntax formats onto the code fragment, we see that the compiler parses it as shown in the picture below, treating the inner `if-else` statement as a single statement being repeated by the outer `for` statement.

```
for ( int c=0; c<input.length(); c++ )
```

```
if ( Character.isLetter( input.charAt( c ) ) )
```

```
letterCnt++;
```

```
else
```

```
nonLetterCnt++;
```

Thus, the `for` statement repeats the if-else statement for \( c = 0, 1, 2, \ldots \) up to the length of the string `input`. The method call to `input.charAt` returns the character within the string at the given position and the call to `Character.isLetter` returns true if the character is a letter of the English alphabet. Thus, the code fragment counts the number of letters and non-letters within the `String` object `input`. 