[0360] C# Iteration

Time to talk about C#'s Flow of Control statements - the keywords that tell the computer what to do next. The good news is that C#'s Flow of Control statements are VERY similar to Java's. The bad news is that you need to watch out for the slight differences that do exist or you will get very confused.

C# Iteration Statements

Just like with Java, C# has 4 iteration (i.e., Looping) techniques:

1. for Loops (Old Style)
2. foreach/in Loops (for Collections)
3. while/do Loops
4. do/while Loops

for Loops (Old-Style)

These are the classic for loops that include an loop counter. These type of loops are used when you need to execute your loop a fixed number of times and you know that number before entering the loop.

```
for (int i=0; i < 33; i++)
{
   statements;
}
```

Old-style for loops should always be written like this. While it is possible to get "creative" with the three parts of the statement you should never, ever do so. Some other notes:

- Use well-known temporary variable names for your loop counter - i, j, k or possibly x, y, z.
- Whenever possible, count up from 0 incrementing by 1. Also use "i < Limit" as the loop condition, where Limit is a constant.

If you find yourself changing your for loop significantly from the example above, consider using a different kind of loop statement.

foreach/in Loops (for Collections)

If you want to do something to every item in a Collection, use the foreach/in style loop instead of the old style version.

```
foreach (string name in studentNames)
{
   statements;
}
```

Remember that ***arrays are Collections*** and foreach is perfect for array operations (unless you need to know the index of each array item, in which case you'll need to use the old-style version).

while/do Loops and do/while Loops

A while/do loop has its condition at the top of the loop. A do/while loop has its condition at the bottom of the loop. Which one should you use when? Remember this: **If it is possible that under certain conditions your program won't need to execute the code inside a loop at all, you need a while/do loop. Otherwise, you'll need a do/while loop.**

```
while (notComplete)
{
   statements;
}
```

The statements inside that loop will only execute if notComplete is true before the loop starts. If notComplete is false, the body of the loop will be skipped over.

```
do
{
   statements;
} while (notComplete);
```

The statements inside this loop will always execute at least once regardless of the value of notComplete at the start of the loop.