Department of Computer Science

08101 Programming I

Week 3 Laboratory 2013/2014

Before you go onto this lab, please make sure that you have sorted out the Cinema Entry program from last week. This lab builds upon the program, and you need to have a working entry program to start form when you want to do this part.

Repeating Behaviours

We can now make simple programs that read in some data (film number and age), do something with it (decide if you can see the film or not) and display a result (print the message). Actually, this is the only thing that computers ever do, so we are well on the way to becoming proper programmers at this point.

However, there is something that we need to be able to do before we can apply for a job at Google. We need to understand loops.

We have noticed that sometimes our program wants to repeat a behaviour, either because it didn't work first time (the user entered a film number that is not valid) or because we want to do something more than once (perhaps we want to deal with another customer). To do this we need a program construction that lets us repeat a set of C# program statements multiple times (perhaps we want to handle 100 customers), or continue performing a statement until some condition is met (perhaps we want to keep looping while the user gives is stupid values).

Validating inputs

Programmers call it "validation". We call it "making sure you don't get stupid values from your user". In the case of the cinema entry program you will somewhere have to deal with the user putting in stupid numbers in two different situations, giving the film number and giving their age. In both these situations we want the program to keep on asking for a numeric value while the user gives them invalid ones.

Validating the Film Number

In the case of the film number we will have something like this, where what the user types is underlined:

```
Welcome to our Multiplex
We are presently showing:
    1. Rush (15)
    2. How I Live Now (15)
    3. Thor: The Dark World (12A)
    4. Filth (18)
    5. Planes (U)
Enter the number of the film you wish to see: 1
```

If the user makes a mistake you might find something like this:

```
Enter the number of the film you wish to see: 10
```

There is no film with the number 10, and so we want to reject this. In fact, if the user gives a film number which is less than 1 or greater than 10 we want to reject this. It turns out that we can combine these two tests into a single condition, using a *logical or*.

```
if ((filmNumber < 1) || (filmNumber > 5))
{
    // if we get here the age is wrong
}
else
{
   // if we get here the age is OK
}
```

You can find out more about the if condition and the tests you can do in the C# Yellow Book on page 39.

Remember that if you use this code you need to use the variable you created to hold the film number (I called it `filmNumber`) in your code.

Making Loops to Validate the film number

You can use the above code to make an easy test for the age, but what you really want to do is loop around if the age is wrong. You want to **do** the read operation **while** the number given is wrong. To do this we can use the rather aptly named do-while loop.

```
do
{
    // read in the value of filmNumber
} while ((filmNumber < 1) || (filmNumber > 5));

// When we get here we must have a valid film number
```

This loop will go round until the user types in a valid film number value. In other words, as long as the condition is true, this loop will repeat.

You can find out more about loops in the C# Yellow Book on page 43.

Now that we know about loops we can modify our program to repeatedly request film number values until the user enters a valid one.

Before you go any further; perform the following:

1. Modify the **`Cinema.cs`** program so that it refuses to accept a film number which is outside the range 1 to 5.
2. Use the same technique to ensure that ages are entered in the range 5 to 120. These are the minimum and maximum age values that the cinema manager wants you to use.

Making your program user friendly

Ideally your program should give messages to tell the user that they have done something wrong:

```
Welcome to our Multiplex
We are presently showing:
    1. Rush (15)
    2. How I Live Now (15)
    3. Thor: The Dark World (12A)
    4. Filth (18)
    5. Planes (U)
Enter the number of the film you wish to see: 10
That film number is invalid.
Enter the number of the film you wish to see:
```

It would be nice if you could improve your program to do this.

Before you go any further; perform the following:

3. Modify the **`Cinema.cs`** program so that it produces an error message if the user gives a value which is out of range. One way to do this is to repeat the test in the program. The first test produces the message if the input is invalid and the other test controls the loop.
4. If you are feeling ambitious you can take a look at the **`break`** keyword and see if you can use this to make a version which only does the test once. You can find the **`break`** keyword on page 46 in the Yellow Book

Putting a Loop inside a Loop

The program we are writing should actually run continuously, in other words it should not stop after it has run once. We can get this effect by putting a loop around the entire program which will repeat it until the user says stop:

```
Another customer? (Y or N): Y
Welcome to our Multiplex
...
```

This version of the program asks the user if they want to process another customer. If the answer is Y the program will go round again. The program layout will look a bit like this:

```
string doWeGoRoundAgain;

do
{
    // code to process one film customer

   Console.Write("Another customer? (Y or N):");
   doWeGoRoundAgain = Console.ReadLine();
}
while (doWeGoRoundAgain == "Y");
```

It is perfectly OK for us to put loops inside loops, but remember to use indenting to make it clear what is happening.

Before you go any further; perform the following:

5. Modify the **`Cinema.cs`** program so that it repeatedly processes customers until the user says stop.
6. If you are feeling ambitious you can take a look at the **`string.ToUpper()`** method that returns an upper case version of a string. This would allow you to write a program that works whether the user types Y or y.

Advanced Stuff – Handling Exceptions

The program you have written is nearly perfect, but not quite. If you are felling really ambitious you can add exception handling to the program, so that the program behaves correctly if the user does something like this:

```
Enter the number of the film you wish to see: bang
```

This problem is caused by the Parse method getting upset and throwing an exception. You can find out how to deal with this by looking at page 65 in the C# Yellow Book.

eBridge Lab Summary

Don't forget to go onto eBridge and do the summary test for this lab.

Rob Miles

October 2013