Mr. Bellavia

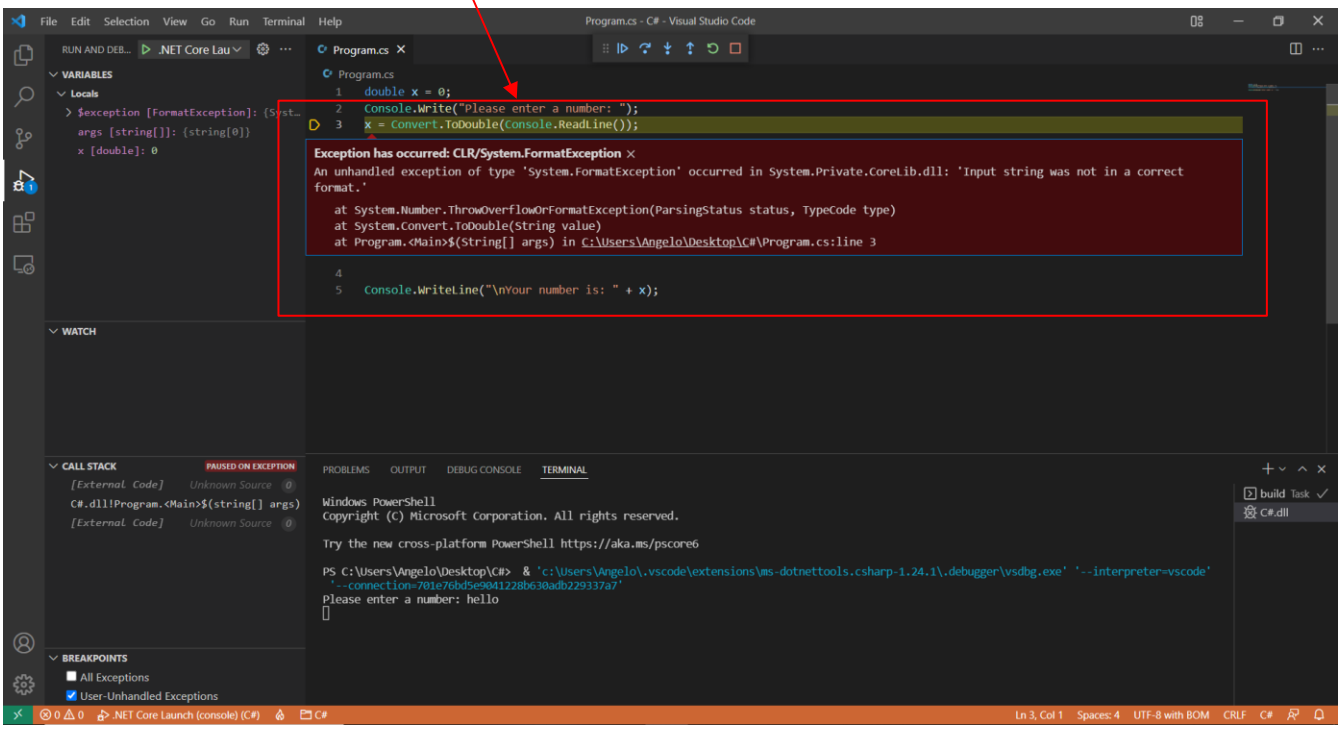
LESSON 05 - TRY-CATCH-STATEMENTS & BOOLEAN VARIABLES

In this lesson we will examine how to handle runtime errors (exceptions) and explore the use of Boolean variables.

Sections:

I. EXCEPTIONS:

We have encountered **exceptions** already in our programs. For example, when a user is expected to type a number, but they type a string instead:



The above screenshot shows what an exception looks like in Visual Studio (not pretty). This exception is a runtime error in which the code encounters an error (exception) while it is running. Once this happens the program crashes! When this occurs, we say that our program has **Thrown an Exception.** Let's breakdown what happened above:

```
double x = 0;
Console.Write("Please enter a number: ");
x = Convert.ToDouble(Console.ReadLine());

Console.WriteLine("\nYour number is: " + x);
```

Sample **Input** & **Output**:

```
Please enter a number: hello
```

Program crashes! **(Throws an Exception)**

The program throws an exception when it tries to convert the inputted text 'hello' to a **double.** This is not allowed! Up until this point, we have assumed that the user would never type a string if our program was asking for a number. However, we cannot guarantee that that will be the case in real life. This brings us to our next section on how to handle exceptions using the **try-catch-statement.**

Mr. Bellavia

II. TRY-CATCH-STATEMENTS:

When dealing with runtime exceptions we can use a special statement called the **try-catch-statement** to **handle an exception** when it occurs. Consider the following code:

```
double x = 0;

try
{
    Console.Write("Please enter a number: ");
    x = Convert.ToDouble(Console.ReadLine());
}
catch
{
    Console.WriteLine("\nYou did not enter a valid number!");
    Environment.Exit(0);
}

Console.WriteLine("\nYour number is: " + x);
```

Sample **Input** & **Output**:

```
Please enter a number: hello

You did not enter a valid number!
```

Sample **Input** & **Output**:

```
Please enter a number: 4

Your number is: 4
```

Notice how we have surrounded our conversion code with a **try { }** block. We have the keyword **'try'** followed by **braces { }**. Inside these braces, our code attempts to convert the user input to a decimal**.** If an exception occurs inside these braces then our program will **jump** to the **catch { }** block. Here we have the keyword **'catch'** followed by **braces { }**. Inside these braces, our code displays a message telling the user they did not enter a number and then exits the program. If an exception does **not** occur in our try { } block, then the catch { } block will **not** execute.

This demonstrates the basic approach of error handeling. If you anticipate that your code will throw an exception, then surround that piece of code with a try { } block and then use a cath { } block to run code if the exception occurs. If no exception occurs, then our code will simply continue on.

There is much more to exceptions that we can explore (for example, exception types), but for now we are going to stick with this basic format.

III. BOOLEAN VARIABLES:

In this last section, our example showed how to catch an exception and display an apporpirate message. Our program would then exit with the **Environment.Exit(0)** statement. It is more preferred to **not** exit a program like this. Therefore, we must have a way to determine if our code should continue or not based on if the user inputted valid data. This brings us to the **boolean variable.**

The **boolean variable** is simply a variable with a data type of '**bool',** where the value is either **'true'** or **'false',** for example:

```
bool myBool = false;
Console.WriteLine("'myBool' equals: " + myBool);
```

Sample **Output**:

```
'myBool' equals: False
```

Notice the keyword **'bool'** when we declare our boolean variable. Also notice the value **'false'** assigned to this variable. **Note:** When boolean variables are outputted, C# will capitalize the first letter , but in our code the value of the boolean variable is always lowercase.

What makes boolean variables especially useful is our ability to use them as conditions for our if-statements. Let's modify our code from the previous section where we implemented the try-catch:

```
bool validInput = true;
double x = 0;

try
{
    Console.Write("Please enter a number: ");
    x = Convert.ToDouble(Console.ReadLine());
}
catch
{
    validInput = false;
}

if(validInput == true)
    Console.WriteLine("\nYour number is: " + x);
else
    Console.WriteLine("\nYou did not enter a valid number!");
```

Now we are using a boolean variable named **'validInput'** which we initilaly set to **'true'.** If the user does not enter a valid number (i.e., an exception occurs), the try-catch will capture the exception and set **'validInput'** to **'false'.** Finally, we test our **'validInput'** varible using a an **if-statement** and output to the user an appropriate message.

When using boolean variables in if-statements we can use a shortform. For example:

```
if(validInput)
    Console.WriteLine("\nYour number is: " + x);
else
    Console.WriteLine("\nYou did not enter a valid number!");
```

Notice that we did **not** put ' == true ' in our condition above. The above code acts the same way as before, it's just a shortform.

Similarly, we can test for 'false':

```
if(validInput == true)
    Console.WriteLine("\nYou did not enter a valid number!");
else
    Console.WriteLine("\nYour number is: " + x);
```

And the shortform would be:

```
if(!validInput)
    Console.WriteLine("\nYou did not enter a valid number!");
else
    Console.WriteLine("\nYour number is: " + x);
```

Notice the use of the **not operator '!'.** By putting the not operator **'!'** in front of a boolean value you are testing for **'not true'.** All these approaches are valid and you may see a lot of sample code using these shortform methods.