Handling Double Input Exceptions

After reading the blog on Errors and Exceptions, write a C# program that reads and handles double input from the user. Use a *try-catch* block to handle any exceptions that may occur when converting the user's input to double values.

Specifically, handle *FormatException* and *OverflowException* exceptions. If an exception is caught, display an appropriate error message and ask the user to re-enter the double. Repeat this process until the user enters a valid value.

Requirements:

- Use a *try-catch* block to handle *FormatException* and *OverflowException*.
- Display specific error messages for each type of exception.
- Provide a "Try again" option after displaying an error message.
- Use a loop to allow the user to retry.
- Include a *finally* block to ensure that a message is displayed regardless of whether an exception occurs or not.

Example Output:

Enter a double: 12.34
You entered: 12.34
This code always gets executed, regardless of exceptions.
Exiting the Program

Enter a double: abc
Format Exception: The input string ‘abc’ was not in the correct format. Please try again.
This code always gets executed, regardless of exceptions.
Enter a double:

Enter a double: 1.7976931348623157E+400
Overflow Exception: Arithmetic operation resulted in an overflow. Please try again.
This code always gets executed, regardless of exceptions.
Enter a double:

Are you able to provide input to generate an OverflowException? Please check the documentation of Convert.ToDouble() here.

Solution

```
namespace ExceptionExample
{
   public class Program
   {
      public static void Main()
      {
         bool isValidInput = false;
         while (!isValidInput)
         {
            Console.Write("Enter a double: ");
            string userInput = Console.ReadLine();
            try
            {
               double number = Convert.ToDouble(userInput);
               Console.WriteLine("You entered: " + number);
               isValidInput = true; // Set to true to exit the loop on valid input
            }
            catch (FormatException ex)
            {
               Console.WriteLine("Format Exception: " + ex.Message + " Please try again.");
            }
            catch (OverflowException ex)
            {
               Console.WriteLine("Overflow Exception: " + ex.Message + " Please try again.");
            }
            catch (Exception)
            {
               Console.WriteLine("An unexpected error occurred. Please try again.");
            }
            finally
            {
               Console.WriteLine("This code always gets executed, regardless of exceptions.");
            }
         }

         Console.WriteLine("Exiting the Program");
      }
   }
}
```

This C# program is a good example of handling errors and making sure a user provides the correct type of input. A comprehensive breakdown of how the program works is presented below. This explanation was created using ChatGPT. ChatGPT is a powerful tool for understanding how a program operates in-depth, and it is very useful for learning; however, it is not intended for completing your assignments.

- The program starts by defining a namespace called "ExceptionExample" and a class named "Program."
- Inside the "Main" method, it initialises a boolean variable "isValidInput" as "false." This variable is used to control a loop that continues until valid input is provided.
- Within the loop, the program prompts the user to enter a double value and reads the user's input from the console.
- It uses a "try" block to attempt to convert the user's input (a string) into a double using the `Convert.ToDouble` method.

- If the conversion is successful, it prints the entered number and sets "isValidInput" to "true," which breaks the loop as valid input has been received.
- If an exception is thrown during the conversion attempt, the program handles it in one of the following ways based on the type of exception:
    - If a "FormatException" occurs, it prints an error message indicating a format issue and asks the user to try again.
    - If an "OverflowException" occurs, it prints an error message indicating an overflow issue and asks the user to try again.
    - If any other type of exception occurs, it catches it using a general "Exception" catch block, prints a generic error message, and asks the user to try again.
- In all cases, whether an exception is thrown or not, the "finally" block is executed, which prints a message indicating that this code always runs, regardless of exceptions.
- The program repeats this input-validation loop until valid input is provided, at which point it exits the loop.
- After exiting the loop, it prints a message indicating that the program is ending.

In summary, this program continuously prompts the user for a double value, handles various exceptions that may occur during the input conversion, and ensures that the program continues until valid input is received before ending. The "finally" block is used to provide any potential cleanup or finalisation code that needs to be executed regardless of exceptions.

According to the online C# documentation of Convert.ToDouble(), when the method is invoked it can raise either *FormatException* or *OverflowException* depending on the received input. However, further tests of the method showed that *OverflowException* is never thrown by versions of .NET older than 2.2. This is explained in the documentation page of the method Parse, on which Convert.ToDouble is based. Specifically, the *Remarks* section states the following:

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure (exception).

If the above program is tested, e.g., with an input value larger than Double.MaxValue, no *OverflowException* is generated and the Double.PositiveInfinity constant is stored instead. Nonetheless, conversions from *string* to *int* may still produce overflow exceptions that need to be properly handled, as explained in the blog.