Exception Handling Lab

In this lab, we'll exercise exception handling in our projects. Note that for all of these things, you should create tests first!

Let's start off by detecting some business logic errors.

OrderDetail Quantity

1. Open or create a unit test class that covers the ordering process. (OrderTests or something like that).
2. Create a test that expects an Exception. Call it Setting_a_quantity_less_than_0_throws()
3. In that test, instantiate an order detail and set it's quantity property to -1.
4. In the very next line go

```
Assert.Fail("Should never get here. Quantity can't be less than zero");
```

5. Run your test. Got a red light? Good. Let's make it green.
6. Open your OrderDetail class. Create a getter and setter for the quantity property. It is an integer and -1 is certainly an integer, but -1 makes no sense.
7. Create a condition in the setter. If quantity tries to go negative, throw an exception.
8. Re-run and re-code until the unit test turns green.

Customer Id is between one and five characters

If you look at your customer class. Note that its customerId is a string. Weird, I know. But that string is limited to five characters.

9. Open or create a CustomerTests class in your unit tests project.
10. Write IdLongerThanFiveCharsThrowsException(). It should expect an OverflowException.
11. Instantiate a customer and set its CustomerId to something longer than 5 characters.
12. Assert.Fail() on the next line.
13. Run and test. The light should turn red.
14. Create a getter and setter for CustomerId. In the setter, if anyone attempts to set it to six or more characters, throw an OverflowException.
15. Make the test lights turn green.
16. Write a new test called ZeroLengthIdThrowsException(). It should expect an ArgumentNullException.
17. Instantiate a Customer, set CustomerId to "". Assert.Fail() on the next line.
18. Run and test. Got a red light? Good!
19. Go back in your customer class. Add a check, if the CustomerId is empty or null, throw an ArgumentNullException.
20. Run your tests, making sure that all your business rules are being enforced properly.

Catching the exceptions in a UI

Now that you have some good business rules created and enforced, let's practice catching those exceptions in a project with a user interface.

21. Open a Windows Forms project or a Web application project. Which you use is up to you. You may already have one created. If so, feel free to use it.
22. Add a form called "Add customer". In it, provide a textbox for the user to enter a CustomerId. Give them a button to click to add the customer.
23. Behind the button's click, instantiate a customer and set its CustomerId to whatever the user put in the textbox. Then pretend to add the customer. Bring up a message to the user saying "Customer <CustomerId> was successfully added."

24. Run and test. Put in a customer Id that is less than five characters. It should work. Re-test with a customer Id that is more than five characters. Your program should abend miserably!
25. Wrap the logic with a try-catch. If the OverflowException is thrown, let the user know the problem and how to solve it.
26. Run and test again with a zero-length CustomerId. It should fail again.
27. Add a catch for any other generic exceptions. Tell the user "An unknown exception has occurred. Try again."
28. Run and test again with a zero-length CustomerId. This time, it should be caught by the generic exception. But the error message is just too general. Let's add a check for the ArgumentNullException.
29. Add a catch clause to see the ArgumentNullException. Tell the user that the CustomerId can't be empty.
30. Run and test again.

Once you're catching all errors and handling them properly, you can be finished.