TDD & BDD
Design Through Testing

Exercise 2
Dealing With Exceptions

Overview

In this exercise our aim is to learn how to test exceptions.

One of the commonest causes of failure in production systems are mistakes in error-handling code. We need to test our code when things are going wrong, as well as when everything is working as we expect.

This is a big benefit of TDD, because it allows us to fake the errors in the context of a test. This means that we can be a lot more confident in our code when things do go wrong.

Goals of the Exercise

Apply ***Red, Green, Refactor*** to adding an exception to the StringCalculator.

Continuous Delivery Ltd
Registered Company No: 9094073 England & Wales
VAT: GB 190 6295 90

Exercise

Add a feature to the StringCalculator that will generate an exception if you ask it to add a negative number.

Remember to start with a failing test!

Hints

If your language doesn't support exceptions, return an error code instead.

There is almost certainly support for exceptions in your unit test framework if your language supports exceptions. Here are a few examples, if you can't see your language here, google for something like:

"How to test for exceptions in <your language>".

Python:

```
self.assertRaises(MyException, a_broken_function)
```

or

```
with self.assertRaises(MyException) as context:
a_broken_function()
self.assertTrue("Houston, we have a problem" in context.exception)
```

Java (JUnit 4):

```
@Test(expected = MyException.class)
public void shouldFailIfSomethingBadHappens() {
   aBrokenFunction();
}
```

Java (JUnit 5):

```
@Test
public void shouldFailIfSomethingBadHappens() {
    Throwable exception = assertThrows(MyException.class,
                                       () -> aBrokenFunction());
    assertEquals("Houston, we have a problem", exception.getMessage());
}
```

Java (Java 8 or later)

```
@Test
public void shouldFailIfSomethingBadHappens() {
    assertThrown(aBrokenFunction)
            .isInstanceOf(RuntimeException.class)
            .hasMessage("Houston, we have a problem")
            .hasMessageStartingWith("Runtime")
            .hasMessageEndingWith("occurred")
            .hasMessageContaining("exception")
            .hasNoCause();
}
```

Javascript (Jasmine, Jest)

```
expect(() => aBadFunction()).toThrow("Houston, we have a problem”);
```

Javascript(vanilla)

```
assert.throws(() => aBadFunction(), MyException);
```

C#

```
Assert.Throws<MyException>(() => aBadFunction());
```

or

```
var ex = Assert.Throws<MyException>(() => aBadFunction());
Assert.Contains("Houston, we have a problem", ex.Message);
```

C++

```
//Exception handling using ‘Catch’ unit test framework
REQUIRE_THROWS(aBadFunction());
```