Lab - Test Driven Development (TDD)

Introduction

We want you to develop more functionality within the Calculator, so it can add numbers together from a string passed into the Add method.

We'd like you to develop this new functionality using test-driven development, where you write the test for each new requirement (and see it fail) before writing the simplest code possible to make that test pass (then keep repeating the process until all requirements have been satisfied).

We suggest working in a pair, where one person writes the test, then the other person writes the code to make the test pass. Between each test, you could swap roles, so that you both get change to perform both sides of the process.

Starter project

We have provided a starter project for this lab in C#, Java and JavaScript flavours.

Goals

Fulfil each of the following requirements in turn, using the test-driven development process. Each time round the process, pick the next requirement, write a test for it (and see it fail), then write the simplest code possible to make the test pass. In the next round, write a test for the next requirement (and see it fail), then write the simplest code possible to make the new test pass, as well as keeping all the earlier tests passing.

The individual requirements (tests) for the Add method are:

1. Empty string should return zero.
2. Just a number should return the parsed version of the number.
3. Two numbers separated by a comma should be added together.
4. Two numbers separated by a newline character ("\n") should be added together.
5. Three numbers separated by either a comma or a newline, or any combination, should be added together.
6. Negative numbers appearing anywhere in the string should throw an exception with the message "Negative numbers not allowed".

More TDD projects if you have time

Exercise: Shopping basket

1. When new item is added the count of items in the basket is increased by one
2. When the same item is added the count of items in the basket remains the same and the added item quantity is increased by 1
3. Attempting to add an item with the quantity of zero or less should throw ArgumentException
4. Attempting to add an item with the price of less than zero should throw ArgumentException
5. Attempting to remove an item that does not exist should throw ArgumentException
6. Removing a number of the same item which is greater than zero and less than the quantity in the basket should decrease the quantity
7. Removing a number of the same item which is greater than zero and less than the quantity in the basket should decrease the quantity
8. Attempting to remove a number of the same item which is greater than its quantity should throw ArgumentException
9. Removing a number of the same item which is equal to its quantity should remove the item from the basket
10. When a few items are added, the total price must be the sum of each item times its quantity

Exercise – Bank account

1. A user can create a new bank account with an initial deposit amount
2. The account should have a unique account number
3. when a user deposits funds into an account balance is increased by the amount
4. depositing negative money will throw an exception
5. when a user withdraws funds from an account balance is decreased by the amount
6. withdrawing negative money will throw an exception
7. withdrawing money greater than balance will throw an exception
8. all recorded transactions must reflect any deposits withdrawals
9. closing an account with a balance other than zero will throw an exception
10. account is marked an inactive when it is closed