Software and Programming II — 2014/15

Coursework Two

Set: Tuesday, 14th October 2014
Due: Monday, 3rd November 2014 by 23:55

Your `github` repo should be named "`sp2-cw2-2014`".

Aims of this coursework

To improve your knowledge of classes and objects. To allow you to experiment with TDD techniques, especially `JUnit`.

Required

1. Extend the `FractionTest` and `Fraction` classes associated with this coursework assignment.

2. Write a simple text based calculator to compute with fractions.

Suggested approach

The `Fraction` API

You should use Test Driven Design (TDD). You have been provided with a `multiply` method and you should write similar methods for `add`, `subtract`, and `divide`.

Write a method `absValue` to return the absolute value of a fraction, and `negate` to change the sign of a fraction. All of these methods should return a newly created `Fraction`, not change any existing `Fraction` object.

Improve upon the `toString` method so that, when given a `Fraction` whose denominator is 1, `toString` just returns the numerator (as a `String`).

The text interface

Write separate classes, `FractionCalculatorTest`, and `FractionCalculator`. The code described in this section goes in those classes, **not** in `Fraction` or `FractionTest`.

1. When the program first starts, the value in the calculator should be zero.

2. Accept lines of input from the user. Each line will contain some mixture of numbers and operators, separated by spaces, For example,

```
3/4 + 1/-3 * 7 / 5
```

Here's what to do for each input item:

When you see...	Do this...
`+`	Remember (in some variable) that you need to do an addition.
`-`	Remember (in some variable) that you need to do a subtraction.
`*`	Remember (in some variable) that you need to do a multiplication.
`/`	Remember (in some variable) that you need to do a division. (You can tell a division operator from part of a fraction because the `/` is all by itself.)

For each of these, if there is already an operation being remembered, then raise an appropriate exception.

When you see...	Do this...
a or A or abs or...	Take the absolute value of the fraction currently held by the calculator.
n or N or neg or...	Change the sign of the fraction currently held by the calculator.
c or C or clear or...	Set the value held by the calculator to zero.
A fraction	If you have a remembered operation (`+`, `-`, `*`, or `/`), perform the operation, with the value currently in the calculator as the first operand, and the fraction as the second operand. The result becomes the new value in the calculator. "Forget" the operation, since you have now used it. If you do not have a remembered operation, then set the value in the calculator to the new fraction.
A whole number	Treat this as a Fraction whose denominator is 1.
q or Q or quit or...	Raise an exception
Anything else	Stop processing any remaining input, set the value in the calculator to zero, and raise an exception

Extended example:

After this input:		1/2	-	3/4	*	abs	`\n`	8	7/8	neg	+
Value in calculator:	0/1	1/2	1/2	-1/4	-1/4	1/4	1/4	8	7/8	-7/8	-7/8
Remembered operation:	None	None	-	None	*	*	None	None	None	None	+

In `FractionCalculator`, write a method `evaluate(fraction, inputString)`. The arguments are a `Fraction` to use as a starting value, and a `String` such as the user might type in. The result should be a new `Fraction` that is the result of carrying out the arithmetic, or the method might raise an exception that the calling method must catch. This method should do no input or output and **must be unit tested**.

The recommended way of carrying out the unit testing is:

1. Write a test for the feature you are about to add (such as addition, or entering a new fraction, or throw an appropriate exception.
2. Test. This is just to make certain that the new feature fails, and to some extent is "testing the test".
3. Add to the `evaluate` method the code needed to handle the new feature.
4. Test. If the test fails, go back and fix either the code or the test until all the tests pass.
5. If you can see any way to improve the code without changing what it does (this is called "refactoring"), do it now, and test to make sure you haven't broken anything.
6. Repeat for the next feature.

Now, write a `main` method to:

1. Initially set the value in the calculator to zero, with no *remembered* operation.
2. Print some kind of welcome message. It should include your name.
3. Read lines of input from the user and, for each line, print just the final result of calculating that line.
4. Leave this final result as the current value in the calculator, so that the user can continue on the next line.
5. If any kind of exception occurs (except the end of input), print the word `"Error"`, reset the calculator to its initial state, and discard the remainder of the input line,
6. For an *end of input* exception just print the word `"Goodbye"` and exit the program.

Submission

We will clone your `github` repo at the due date and time.