SE/CS 2S03: Principles of Programming

Due on December 4th

Dr. Jacques Carette

1 Goals

The goals of this assignment are:

1. get more familiar with OO programming with lots of classes.
2. put a nice front-end on it.

2 The Task

The *primary* task is to implement the "core" of a calculator application which can deal with both integer and floating point numbers. A *secondary* task will be to hook that up to a GUI frontend.

The core functionality, done as a class hierarchy acting as the backend to a String parser, will be worth 80%, while hooking up the GUI is 20%.

The calculator application will include add, subtract, multiply and divide of *expressions*. It will work in two modes: integer and floating point. So `1+3` is `4` in integer mode and `4.0` in floating point. `1/3` is an error in integer mode, and `0.333333` in floating point.

3 The details

3.1 The classes

In lectures, you were shown an `Expr` class and many sub-classes of that class. You will add two methods:

1. `evalToInt` which will evaluate the expression represented by the object to an `int`. If the expression does not denote an integer, throw an Exception `NotAnInteger`.
2. `evalToFloat` which will evaluate the expression represented by the object to a `double`.

Note that Java itself will throw an exception when you divide by zero. Let it (i.e. `1/0` should not thow `NotAnInteger`).
Find a good home for the following code:

```
public enum Mode { INTEGER, FLOAT }

abstract class Value {
    Mode m;
}

class IntVal extends Value {
    int val;
}
```

```
class DblVal extends Value {
    double val;
}
```

3.2 Parsing and Evaluating

You will be provided with a class `Parser` with a single public method `parse`, which returns an `Expr`. For the non-bonus parts of this assignment, you do not need to modify this class.

You need to create a class `Evaluate` with:

1. a single constructor `Evaluate(String s, Mode m)`
2. a method `public Value eval()`
3. exactly two fields, `private Expr e` and `private Mode m`.

Note that your eval method will inevitably throw some exceptions.

3.3 Testing

You need to test your classes, and you need to test that your classes behave properly when hooked up behind the Parser.

1. Create a set of *scenarios* of use of your classes via (hardcoded) sequences of constructor calls to `Expr`. For example `Expr test1 = new Plus(new Integer2(1), new Integer2(2));` and then call each of `evalToInt` and `evalToFloat` and make sure you get the expected answer.
2. Create a set of *scenarios* as (hardcoded) strings. Then use `Evaluate` to make sure they work as expected.

You need to make sure to create scenarios which lead to exceptions being thrown. For the string scenarios, this includes parsing errors.

3.4 GUI

To make things pretty, you will also implement a graphical front-end. The calculator should only display the input as entered, until the '=' key is entered: the result should then be computed and displayed. Provide parentheses as input as well.

For example, the key presses `1`, then `+` then `3` would display `1+3`. Pressing `=` would change the display to `1+3=4`.

You should also implement an `AC` (for *all clear*) button, to start new computations.

For the implementation, you *will*:

- start from the UI code (provided as fora5.zip), minus the action code.
- remove '%' from the UI and replace it with 'B' (for backspace).
- remove '.' from the UI and replace with a mode switcher. Use color to indicate which mode the calculator is in.

As there are a number of corner cases, here is the specification for those cases:

- If the input is syntactically incorrect, then simply display `syntax error`. In other words, pressing `=` when you have previous entered something (like) `1+)` would result in `1+)= syntax error`. Pressing 'backspace' would then revert the display back to the bad input (i.e. remove '= syntax error').
- In integer mode, if the result is not an integer, let the result be (the text) `fraction`. Backspace should work similar to the above. This includes division by zero.

- In floating point mode, division by zero should cause the result to display as `NaN`. infinities should also 'work'.
- Changing the mode only effects what `=` *does*. It otherwise has no effect.

Extremely Important: Nether your GUI code nor your Parser should do **any evaluation**. The **only** method which knows how to evaluate your expressions should be eval.

For example, after entering `1/2` and pressing `=` in float mode should produce `1/2=0.5000000` (where you should print all results to 6 digits of accuracy). In integer mode, the display should be `1/2=fraction`.

Since testing GUIs is a rather complex task to automate, you will instead write a `Testing.txt` file which contains a description of the scenarios (precision matters here!) and expected results.

3.5 Further notes

If you have done everything correctly, there should be at most three conditionals on `mode`, one in `Evaluate` and two in your calculator.

4 Submission Requirements

- A *single* zip file containing all your java files, including all the ones provided (whether modified or not).
- For this assignment, put the files in package `cs2s03`.
- Extra files are ok.

5 Marking Scheme

- Correct `Expr` class hierarchy: 50%.
- Tests for `Expr` class: 15%.
- Tests for `Parser` and `Expr`: 15%.
- Properly modified GUI code: 15%.
- A `Testing.txt` file with instructions for humans to test the GUI: 5%.

6 Bonus

Each one of these will be worth extra marks:

- (easy-medium) Improve the parser to implement operator precedence, and to give better error messages.
- (medium) Have *no* conditionals on mode. Hint: First class functions (see the rosettacode.org entry for first class functions, look up the Java 8 way of doing that on that page) and arrays. By 'conditional' I mean `if`, `?` and `switch`. Each conditional removed will be worth some bonus marks.
- (medium) Find a way to automate the testing of the GUI code.

You may do multiple bonus. Bonus for this assignment are due at the same time as the assignment. Please hand all bonus parts as a sub-directory `bonus` in your zip file.