Lab Exercise 5
CS 2334

September 24, 2015

Introduction

This lab focuses on the use of Exceptions to catch a variety of errors that can occur, allowing your program to take appropriate corrective action. You will implement a simple calculator program that allows the user to specify an operator and up to two operands (parameters). Your program will parse these inputs, perform the operation and print out the result. If an error occurs during any of these steps, your program will catch the errors and provide appropriate feedback to the user.

Learning Objectives

By the end of this laboratory exercise, you should be able to:

1. Create a text menu-based application
2. Implement and throw a custom Exception
3. Robustly handle Exceptions with a try/catch block

Proper Academic Conduct

This lab is to be done individually. Do not look at or discuss solutions with anyone other than the instructor or the TAs. Do not copy or look at specific solutions from the net.

Preparation

1. Create a **lab5** project in your Eclipse workspace.

User Interaction

Your program will begin by providing the user with the instructions:

```
Welcome to the Eclipse calculator!

Please select an option and give the parameters
Addition:       1 x y
Subtraction:    2 x y
Multiplication: 3 x y
Division:       4 x y
Power:          5 x y
Factorial:      6 x
Quit:           7
```

Your program will then wait for the user to enter a line containing the operator number (1–7) and the appropriate number of parameters. After performing the operation and printing the result, your program will wait for the next input.

Notes:

- The operator and parameters all occur on a single line and are separated by space characters.
- Your program must indicate if an error has occurred in specifying the operator or the parameters. If so, an error message is printed and your program returns to waiting for the next line of input. Remember that Exceptions can be created with a *message* String that describes in words the nature of the error.
- It is an error if an operator does not fall into the range 1–7.
- It is an error if operators 1–5 do not have exactly two parameters (these are *binary operators*).
- It is an error if operator 6 does not have one parameter (this is a *unary operator*).
- It is an error if operator 7 has any parameters.
- Your program will exit if operator 7 is selected.

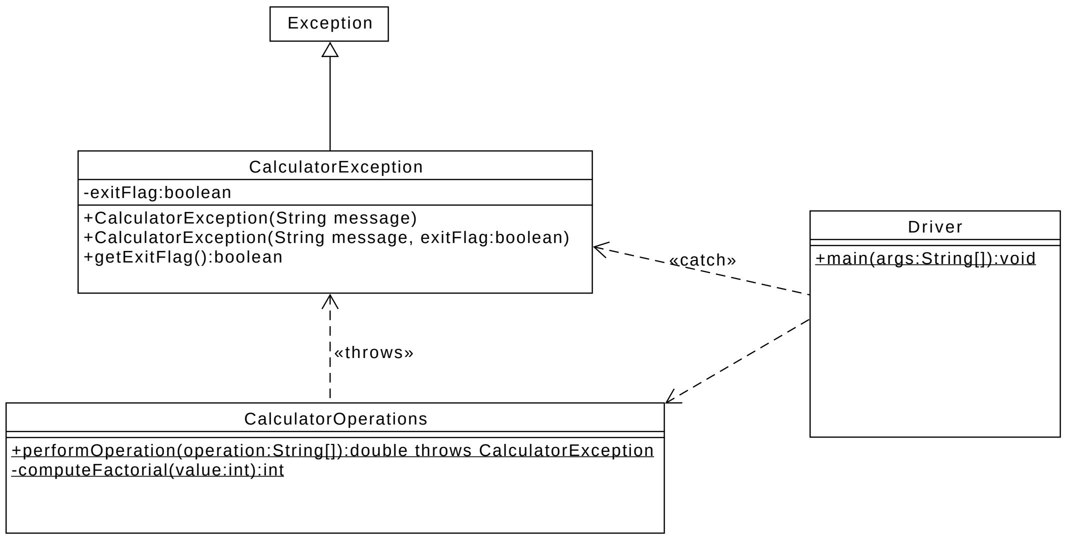
Class Design

Below is the UML representation of the set of classes that you are to implement for this lab (note that the *Exception* class is provided by the Java API).

It is important that you adhere to the instance variables and method names provided in this diagram (we will be executing our own JUnit tests against your code). It is also important that you maintain a clear separation between the *Driver* class and the *CalculatorOperations* class.

The *Driver* class is responsible for receiving the inputs from the user, splitting the operator from the operands, printing the results and catching and printing any errors that might occur.

The *CalculatorOperations* class is responsible for parsing and interpreting the operator and operands and performing the operation. It is this class that is responsible for detecting **any** errors that occur.


Exception
CalculatorException
-exitFlag:boolean
+CalculatorException(String message)
+CalculatorException(String message, exitFlag:boolean)
+getExitFlag():boolean
«catch»
Driver
+main(args:String[]):void
«throws»
CalculatorOperations
+performOperation(operation:String[]):double throws CalculatorException
-computeFactorial(value:int):int

The *CalculatorException* class is derived from *Exception*, and adds an instance variable, called *exitFlag*. This flag is set to true to indicate that the program should terminate.

The line from *CalculatorOperations* class to *CalculatorException* simply indicates that the former may throw the latter under certain conditions. Note that *CalculatorOperations* must only throw exceptions of this type (and no others).

The line from *Driver* to *CalculatorException* indicates that the former might have to catch the latter.

The line from *Driver* to *CalculatorOperations* indicates that the former calls methods provided by the latter.

You must implement your own JUnit test classes called *CalculatorExceptionTest* and *CalculatorOperationsTest*.

Here is a general outline of a possible implementation for performOperation():

```
public double performOperation(String[] operation){
  // Interpret operation[0] as an int

  // If there is at least one parameter, then interpret this
  //  parameter as a double

  // If there is at least a second parameter, then interpret this
  //  parameter as a double

  // Note the total number of parameters

  // Depending on the operation type, execute the operation and
  //  return the result

}
```

Implementation Steps

1. Before you implement the full program, implement the *Driver* and the *CalculatorOperations* classes assuming that the user is perfectly behaved and that no exceptions will be thrown. Get this version of the program working before you move on to the next step.

2. Implement a JUnit test for the *CalculatorOperations* class.

3. Implement the *CalculatorException* class and a corresponding JUnit test.

4. Introduce error handling to the rest of your program. Configure the *CalculatorOperations* class to throw CalculatorException. In addition, configure your *Driver* class to catch CalculatorExceptions. Note that **this is the only Exception that your Driver may catch**. If there are other exceptions that occur inside of CalculatorOperations, then this class must catch and address them (in some cases, this means turning around and throwing a CalculatorException).

   Note that it is okay for your main method to declare that it throws *IOException*.

Final Steps

1. Generate Javadoc using Eclipse.
    - Select *Project/Generate Javadoc...*
    - Make sure that your project is selected, as are the CalculatorException, CalculatorOperations, Driver, CalculatorExceptionTest and CalculatorOperationsTest classes (check for these individually!)
    - Select *Private* visibility
    - Use the default destination folder
    - Click *Finish*
2. Open the *lab5/doc/index.html* file using your favorite web browser or Eclipse (double clicking in the package explorer will open the web page). Check to make sure that that all of your classes are listed and that all of your documented methods have the necessary documentation.
3. If you complete the above instructions during lab, you may have your implementation checked by one of the TAs.

Submission Instructions

- All required components (source code and compiled documentation) are due at 11:59pm on Friday, September 25th.
- Prepare your submission file:
    1. Select the project in the *Package Explorer* window.
    2. Right-click. Select *Export*
    3. Select *General/Archive File*
    4. Expand your project and verify that both the *src* and *doc* folders are selected, as well as all of their contents
    5. Enter the archive file name: *lab5.zip* (note that you may want to browse to a different destination folder)
    6. Select *Save in zip format*
    7. Click *Finish*
- Submit your zip file to the lab5 folder on D2L.

Rubric

The project will be graded out of 100 points. The distribution is as follows:

Implementation: 35 points

Program formatting: 5 points

(5) The program is properly formatted (including indentation, curly brace and semicolon locations).

(3) There is one problem with program formatting.

(0) The program is not properly formatted.

Data types and method calls: 10 points

(15) The program is using proper data types and method calls.

(10) There is one error in data type or method call selection.

(5) There are two errors in data type or method call selection.

(0) There are multiple errors in data type and method call selection.

Required Methods: 10 points

(10) All of the required methods are implemented.

(7) A component of one required method is missing.

(4) A method is not implemented or components are missing from two methods.

(0) Two or more required methods are not implemented or components from three or more methods are missing.

JUnit Tests: 10 points

(10) An appropriate set of JUnit tests is implemented

(7) One JUnit test component is missing

(4) Two JUnit test components are missing

(0) Three or more JUnit test components are missing

Proper Execution: 30 points

Output: 15 points

(15) The program passes all tests.

(10) The program fails one test.

(5) The program fails two tests.

(0) The program fails three or more tests.

Execution: 15 points

(15) The program executes with no errors.

(8) The program executes, but there is one minor error.

(0) The program does not execute.

Documentation and Submission: 35 points

Project Documentation: 5 points

(5) The java file contains all of the required documentation elements at the top of the file.

(3) The java file is missing one of the required documentation elements.

(2) The java file is missing two of the required documentation elements.

(0) The java file is missing more than two of the required documentation elements.

Method-Level Documentation: 10 points

(10) Every method contains all of the required documentation elements ahead of the method prototype.

(7) The method documentation is missing one of the required documentation elements.

(3) The method documentation is missing two of the required documentation elements.

(0) The method documentation is missing more than two of the required documentation elements.

Inline Documentation: 10 points

(10) Every method contains appropriate inline documentation.

(7) There is one missing or incorrect line of inline documentation.

(3) There are two missing or incorrect lines of inline documentation.

(0) There are more than two missing or incorrect lines of inline documentation.

Submission: 10 points

(10) The correct zip file name is used and has the correct contents.

(5) The correct zip file name is used, but one required component is missing.

(0) An incorrect zip file name is used or more than one required component is missing.