Lab Exercise 5: Exceptions
CS 2334

September 22, 2016

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 (arguments). 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 program that interacts with a user through text
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. Download the lab5 partial implementation:
   `http://www.cs.ou.edu/~fagg/classes/cs2334/labs/lab5/lab5.zip`

User Interaction

Each line that is typed by the user is interpreted as a potential expression. Valid expressions consist of a sequence of one, two or three tokens (each token is separated from the preceding token by a space), and may take on one of the following forms:

- 1 token: [**quit**]. The program responds by exiting
- 2 tokens: [**UOP N**], where N is an integer and UOP is a unary operator (-). The program responds by displaying the negative of the given integer
- 3 tokens: [**N1 BOP N2**], where N1 and N2 are integers and BOP is a binary operator (+, -, *, /, or %). The program responds by displaying the result of applying the designated operator to the two arguments

Inputs resulting in an illegal integer operation or not following one of these formats result in the display of a specific error message.

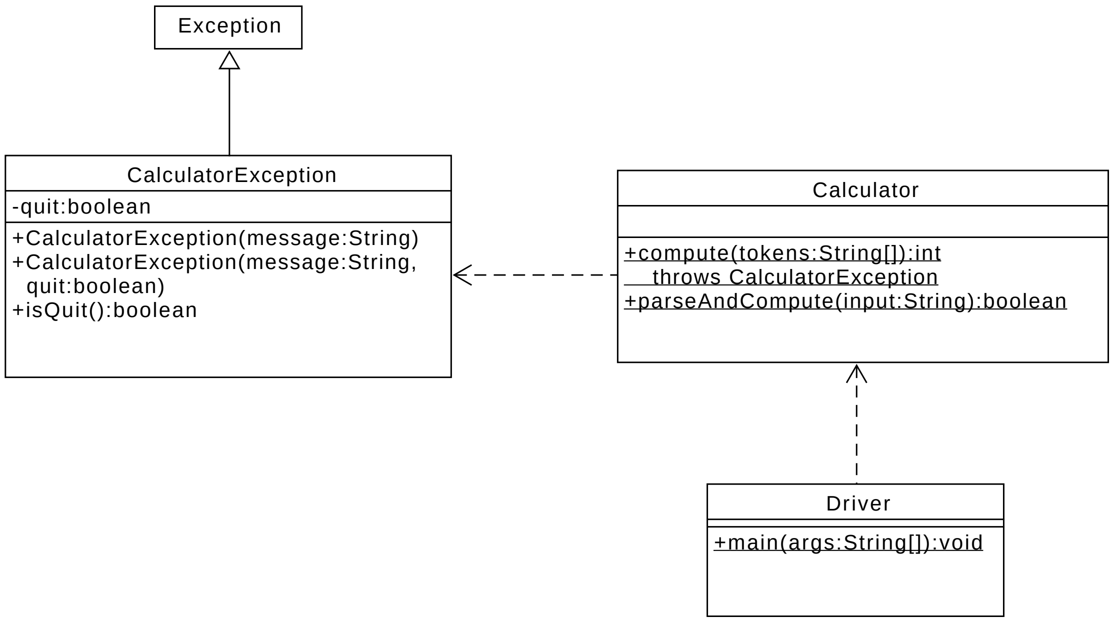
Below is an example interaction with a user. Note that both the user's input and the program's response are shown.

```
4 + 2
The result is: 6
42+7
Illegal input.
4 - 2
The result is: 2
4 / 2
The result is: 2
4 % 2
The result is: 0
foo + 2
Illegal argument.
42 * 3
The result is: 126
42 * bar
Illegal argument.
42 ^ 3
Illegal operator.
32 ^ baz
Illegal argument.
foobar ^ 3
Illegal argument.
4 / 0
Divide by zero error.
42 % 0
Mod by zero error.
-4
Illegal input.
- 4
The result is: -4
QUIT
Exit.
```

Class Design

Below is the UML representation of the set of classes that you are to implement for this lab. 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). In this diagram, you are seeing some new notation: the dashed open arrow means that there is some loose relationship between the classes. It is not an *is-a* relationship (class inheritance), or even a *has-a* relationship (a class or instance variable referring to another class). This relationship is much more nebulous – here, we are acknowledging that one class has local variables that reference another class.

The *Exception* class is provided by the Java API. The *CalculatorException* class is derived from *Exception*, and adds an instance variable, called *quit*. This flag is set to true to indicate that the program should terminate. This class provides two different constructors: one that allows the caller to set the *quit* flag; the other sets the *quit* flag to *false* by default.


Exception
CalculatorException
-quit:boolean
+CalculatorException(message:String)
+CalculatorException(message:String, quit:boolean)
+isQuit():boolean
Calculator
+compute(tokens:String[]):int throws CalculatorException
+parseAndCompute(input:String):boolean
Driver
+main(args:String[]):void

The *Calculator* class provides two methods. The following method is responsible for taking as input a single String that is to be interpreted as an expression.

```
public static boolean parseAndCompute(String input)
```

This method:

1. Separates the String into a set of tokens (substrings that are separated by spaces)
2. Calls *compute()* to evaluate the expression
3. Prints out the result or an error message
4. Returns a boolean to indicate whether the program should terminate

The *compute()* method is responsible for interpreting the set of tokens and producing a result:

```
public static int compute(String[] tokens) throws CalculatorException
```

If there are two or three tokens that make up a valid expression, then this method returns the int result. In all other cases, this method throws a *CalculatorException*, encoding the error String in the *message* property of the *Exception*. When there is exactly one token that is equal to the String "quit" with any casing, the exception's *quit* flag is set to *true*. The details for the appropriate exception message are given in the code skeleton that we provide.

You must implement your own JUnit test class called *CalculatorTest*. We have provided *CalculatorSampleTest* that gives a few hints as to how to test with Exceptions.

The *Driver* class is provided and is responsible for opening an input stream from the user and repeatedly reading and evaluating lines of input until a *quit* has been received.

Implementation Steps

1. Complete the implementation of the *CalculatorException* class.
2. Complete the implementation of the *Calculator* class.
3. Implement a JUnit test for the *Calculator* class.

Final Steps

1. Generate Javadoc using Eclipse.
    - Select *Project/Generate Javadoc...*
    - Make sure that your project is selected, including all of the Java files
    - 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 both of your classes are listed and that all of your documented methods have the necessary documentation.

Submission Instructions

- All required components (source code and compiled documentation) are due at 11:59:00pm on Friday, September 23rd.
- Method 1: Submit through Eclipse
    1. From the *Window* menu, select *Preferences/Configured Assignment.*
    2. Select your project.
    3. From the Project menu, select *Submit Assignment.*
    4. Under *Select the assignment to submit*, select *Lab 5: Exceptions.*
    5. Click *Change Username or Password....* Enter your Web-Cat username and password. Click *OK.* You should only need to do this step once per session.
    6. Click *Finish.*
    7. Your browser should automatically open a Web-Cat page that shows your submission being graded. After a short wait, the page will show a report of your submission. See the main class web page for a link that describes the Web-Cat output.
- Method 2: Submit directly to the Web-Cat server
    1. From the File menu, select *Export.*
    2. Select *Java/JAR File.* Click *Next.*
    3. Select and expand your project folder.
    4. Select your *src* and *doc* folders.
    5. Select *Export Java source files and resources.*
    6. Select an export destination location (e.g., your *Documents* folder/directory). This file should end in *.jar*
    7. Select *Add directory entries.*

8. Click *Finish*.
9. In your web browser, login to the Web-Cat server.
10. Click the *Submit* button.
11. Browse to your jar file.
12. Click the *Upload Submission* button.
13. The next page will give you a list of all files that you are uploading. If you selected the correct jar file, then click the *Confirm* button.
14. Your browser will then open a Web-Cat page that shows your submission being graded. After a short wait, the page will show a report of your submission. See the main class web page for a link that describes the Web-Cat output.

Hints

- The *message* property of *Exception* should be used to encode error messages.
- It is bad coding style for a *catch* statement to catch all *Exceptions* (unless you really mean to catch all exceptions). Instead, you should only catch the specific exceptions that you expect to happen. This way, other, unexpected exceptions will still result in a halt of your program, making it easier to track down problems.

Rubric

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

Correctness/Testing: 45 points

The Web-Cat server will grade this automatically upon submission. Your code will be compiled against a set of tests (called *Unit Tests*). These unit tests will not be visible to you, but the Web-Cat server will inform you as to which tests your code passed/failed. This grade component is proportional to the fraction of tests that your code passes (so 22.5 points means that your code passed half of the tests)

Style/Coding: 20 points

The Web-Cat server will grade this automatically upon submission. Every violation of the *Program Formatting* standard described in Lab 1 will result in a subtraction of a small number of points (usually two points). Looking at your submission report on the Web-Cat server, you will be able to see a notation for each violation that describes the nature of the problem and the number of subtracted points.

Design/Readability: 35 points

This element will be assessed by a grader (typically sometime after the lab deadline). Any *errors* in your program will be noted in the code stored on the Web-Cat server, and two points will be deducted for each. Possible errors include:

- Non-descriptive or inappropriate project- or method-level documentation
- Missing or inappropriate inline documentation
- Inappropriate choice of variable or method names
- Inefficient implementation of an algorithm
- Incorrect implementation of an algorithm
- Incomplete coverage of your Unit Tests. We expect that your unit tests will test all lines of your code

If you do not submit compiled Javadoc for your lab, 5 points will be deducted from this part of your score.

Note that the grader may also give *warnings* or other feedback. Although no points will be deducted, the issues should be addressed in future submissions(where points may be deducted).

Bonus: up to 5 points

You will earn one bonus point for every two hours that your assignment is submitted early.

Penalties: up to 100 points

You will lose ten points for every minute that your assignment is submitted late.