Lab: Exceptions
CSC 207, “Algorithms and Object-Oriented Design”
Department of Computer Science
Grinnell College
September 16, 2019

1. Recover your code for the Averager class from the Objects and Classes lab. and make sure that the class definition can be compiled.

2. Revise the definition of the accessor method that reports the arithmetic mean of the addends so far provided so that, instead of printing an error report and returning 0 if the addend count is 0, the accessor method constructs a new ArithmeticException object and throws it. Recompile the definition of the Averager class.

3. If you wrote a test program for the Averager class as part of the Objects and Classes lab, recover it and revise so that it creates an averager and immediately invokes the averager’s accessor method, causing it to throw an ArithmeticException. How does our implementation of the Java virtual machine respond when this exception is never caught?

4. Revise the test program so that the critical parts of the code you wrote for the previous exercise are in a try-block with a catch-clause that catches the exception and reports it to the user through System.err instead of allowing the program to terminate abnormally. Recompile and execute the test program to make sure that the exception is caught correctly.

5. The last exercise of the Objects and Classes lab invited you to design and implement additional methods to extend the functionality of the Averager class. One of the suggested extensions was an additional accessor method, maximum, that returned the greatest of the addends entered since the most recent reset operation (or since the creation of the object, if no reset operations have yet been executed). Write (or adapt) an implementation of the maximum method that throws an ArithmeticException if the addend count is 0 when the method is invoked. Add a test to your test program that causes maximum to throw this exception, catches it, and prints an appropriate error report to System.err.

6. The ArithmeticException class has two constructors, one that takes no arguments and another that takes a String. Revise the definition of Averager so that one of the throws-statements that you have added to it throws an ArithmeticException constructed with the zero-argument constructor and the other throws an ArithmeticException constructed with the one-argument constructor. In the latter case, use a string argument that indicates the precise cause and nature of the error. Recompile the class definition and re-run the test program. How, if at all, does the difference between the ArithmeticException objects show up in the output from the test program?

7. The ArithmeticException class also has an accessor method, getMessage, that enables the programmer to extract a String from an ArithmeticException object. If the object was constructed using the one-argument version of the constructor, the recovered string is the one that was provided as the argument.

Revise your class definition and test program so that they use this getMessage facility to construct error reports that uniquely identify the statement in which each exception is thrown.

8. Add a finally-clause to each of the try-statements in your test program. In this clause, print an advisory message reporting that the testing of the code in the try-block has concluded (whether or not an exception was thrown during the test).
I am indebted to my colleague, John David Stone, for assistance with this handout.

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit

http://creativecommons.org/licenses/by-sa/4.0/deed.en_US

or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.