# Lab 04 -- Exception Handling

# Summary

Robust programs require the programmer anticipating issues and handling those issues as gracefully as possible. In this lab you will work through several sample programs to gain familiarity with Java’s exception handling capabilities.

You should incorporate error and exception handling in all of your programming from now on.

# Lab

## Exercise 1:

Complete ExceptionDemo.java on pg. 675. Make the following modifications, once you understand how the given code works:

1. Throw an exception if the user enters less than one donut. Present the user with an appropriate message.
2. The program currently aborts if the user enters a String, instead of an int. Alter the program to process String input and gracefully exit if the user enters input that is not numberic.
   1. Use the Integer Wrapper class, and Integer.parseInt(String s).
   2. You will need to handle the potential NumberFormatException that may be thown by the parseInt method. Keep the original Exception catch block, as this is considered good practice.
   3. Note, many programming languages take user input as a String and then attempt to convert it to the desired data type.

## Exercise 2:

Complete DivideByZeroDemo.java on pg. 686. Make the following modifications and answer the following questions, once you understand how the given code works:

1. Why does your custom exception DivideByZeroException have two constructors?

DivideByZeroException has two constructors so the Exception can contain a default message or a custom set message.

1. Why is the output -- “End of program.”-- not included in the try-catch block?

“End of program.” is not included in the try-catch block so it is guaranteed that the line will execute.

1. Alter the program to use the other DivideByZeroException constructor.
2. Alter the program to accept String input, as it currently abends if the user enters non-numeric data.
   1. Use the Integer wrapper class and handle any possible exceptions.
   2. If the user enters non-numeric data, alert them, and give them the opportunity to try again.
      1. Give them the opportunity to try again, even if they enter non-numeric data in giveSecondChance().

## Exercise 3:

Complete DoDivision.java on pg. 697. Make the following modifications and answer the following questions, once you understand how the given code works:

1. It is often a good idea to throw an exception from a method, and catch it at a higher-level in the program, like a GUI. Why do you think delaying the catching of an exception can be a good design paradigm?

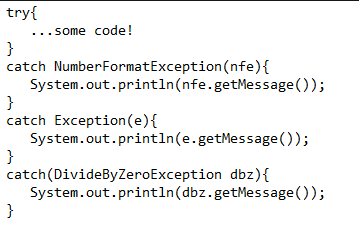
Catching an exception at the GUI layer is a good design paradigm because it gives you a choice on how to

handle the error at the GUI layer rather than just being stuck with how the error is handled at a lower layer.

1. Alter the program to accept String input, as it currently abends if the user enters non-numeric data.
   1. doNormalCase() and giveSecondChance() should throw NumberFormatException and it should be caught in main()
   2. Give the user a second chance if they enter non-numeric data.
2. If the user provides input that is invalid two times, you don’t need to offer them a second chance.

## Exercise 4:

1. A colleague approaches you for assistance with the following code. She mentions it is not working as expected. What suggestion would you provide?



I would suggest ordering the catch blocks starting with the most specific exception and ending with the most generic. In this case, the catch(Exception e){} should be moved to the bottom of the catch block statements.

1. We didn’t use the ***finally*** block in any of the labs above. Research how the ***finally*** block works and provide a brief example of how you might use it.

The finally block would contain any statements that should be executed whether an exception occurs or doesn’t occur. The finally block would be a good place to close input/output streams, close a db connection etc.