CS: 114 **OPPDA** **Spring, 2025**

Lab3 (50 points) **Due 3/31/2025**

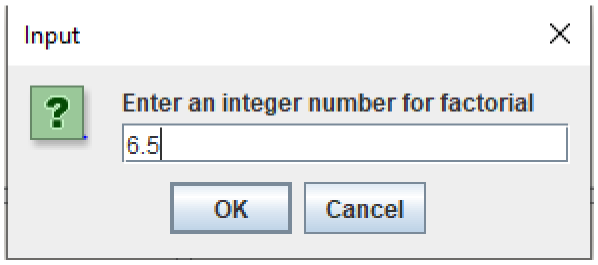
Submit your solutions to canvas. For programming assignments do not send the entire project. All that is needed are the files ending in .java. Each class will have to be defined in its own separate .java file. All driver code should be put in one .java file. Please make sure your name is included at the top of each .java file.

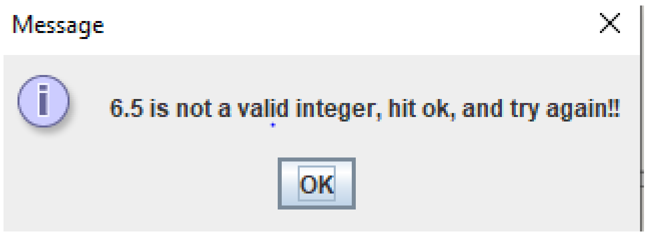
There will be no re-submission of work. There will be no exceptions. So, before you submit your work please ensure you have everything the way you want it.

Please create one driver .java file that calls the methods P1, P2, and P3 for each problem below.

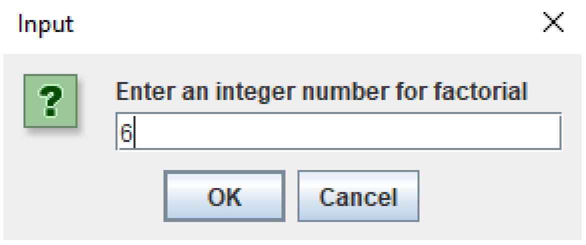
<u>Problem 1</u>

Use the `try-catch` java construct. Use an `JOptionPane.showInputDialog` panel to prompt the user for an integer number to compute the factorial of that number. If the `String` reference variable returned from the `dialogBox` panel is a valid integer via the call to `Integer.parseInt()` then the code should make a call the method `Factorial` to compute the factorial. If the `String` reference variable is not a valid integer the program needs to catch the `NumberFormatException` exception, and have the program user re-enter a valid integer. For example, see the sequence of panels below:



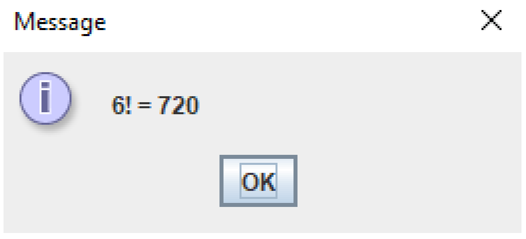

Message

6.5 is not a valid integer, hit ok, and try again!!



Enter an integer number for factorial

6



6! = 720

Additionally, do not allow negative integer numbers. The factorial of 0 is 1. Do your best to prevent overflow for larger numbers. **Hint:** maybe do the computation in something other than an int. **Hint2**: You may want to make use of the Java wrapper classes such as `Double`, `Integer`, `String`, etc.

Problem 2

Implement the same factorial problem in P1, but instead use the console window, a Scanner object, and `nextInt()`, or something similar.

Problem 3

Use the file browser to select a file to open. The file is supposed to just be a file of integers; however, the file may also contain decimal numbers, or strings, or blank lines. It is also possible that the file will be empty or not contain any integer numbers. Do your best to build a robust program to read a file that may contain integers. Read those integers into your favorite Java Collections data structure and remove any duplicates. Next, sort the integers form least to greatest. Finally output the sorted numbers to a file.

If you process the entire file and do not find any integers do not write an output file but instead use a message dialog box to display some sort of message stating, no integers were present in the file.

The goal in all the problems above is to prevent your program from crashing no matter what the user inputs or no matter what file is given for input. Make your program as robust as you possibly can.