ICS 141

Lab #9

Problem Description:

Your goal in this lab is to exercise your ability to use exception handling.

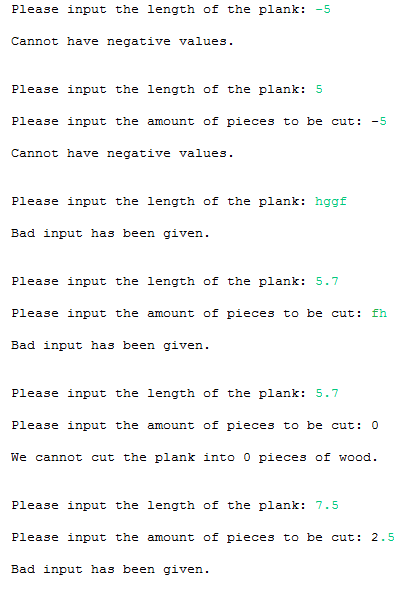
The goal in this program is to cut a plank of wood into a certain amount of equally-long pieces. The program should prompt the user for a double value and an integer value. The double value will hold the length of the plank of wood. The integer value will hold the amount of pieces to be cut. Your goal is to display the amount of pieces cut and the length of each piece. Create and use a simple Scanner object to gather input.

As we know, there are some things that could go wrong here, depending on the user’s input. We could have a divide-by-zero error. We could have a situation where the user inputs a negative length for the original plank. Similarly, the user might attempt to input a negative value for the amount of pieces to be cut. Finally, the user could also input gibberish when the program is expecting a double or int value. You must handle all of these types of exceptions in your program.

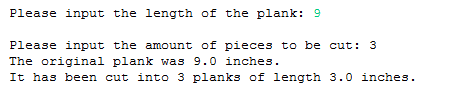
**Hints**

**Since the length of the board is a double value, Java will not actually throw an ArithmeticException when we try to divide by 0. In Java, it is okay to divide a double value by 0 (It’ll only throw that exception if we divide an integer by 0). However, this is still an issue as we’ll get bad answers if the user inputs 0 for the amount of pieces to be cut. Therefore, you should manually create and throw an ArithmeticException in this case (Similar to the example we saw in the slides).**

**Here are a few runs to show how the program should run. Note that this is many executions of the same program. You do not need to loop back to re-accept input.**



**Finally, here is a good run to display how it should appear when all input is okay.**

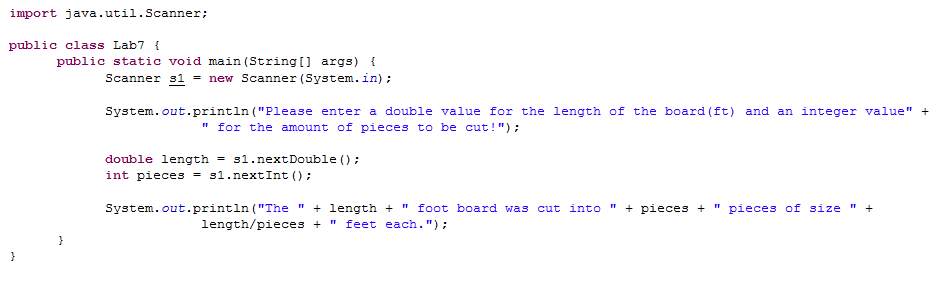


**Deliverables:**

The .java file(s) for this lab. These are the files that contain your source code. I do not need the .class files. Submit the java file(s) to the lab #9 drop box.

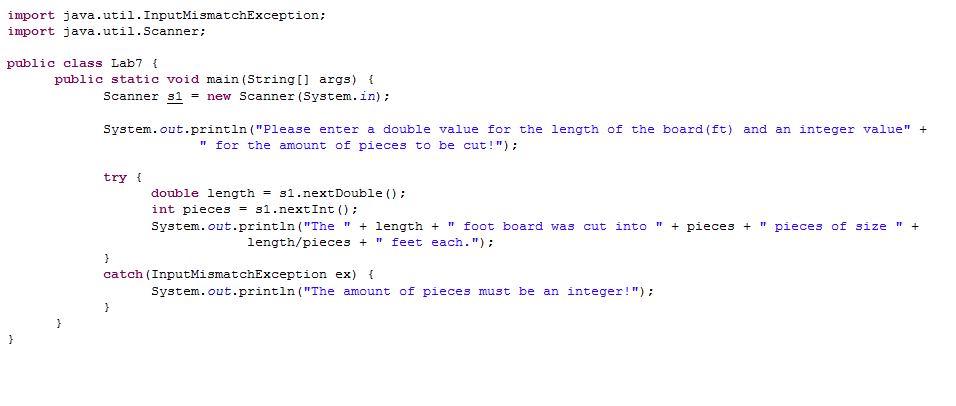
**Walkthrough**

We should be able to contain this entire solution within a single class as we are simply accepting input, making sure it passes our exception handling, and printing output at the end. Let us start by actually solving the problem, and then we’ll go back and introduce the exception handling to make the solution robust.

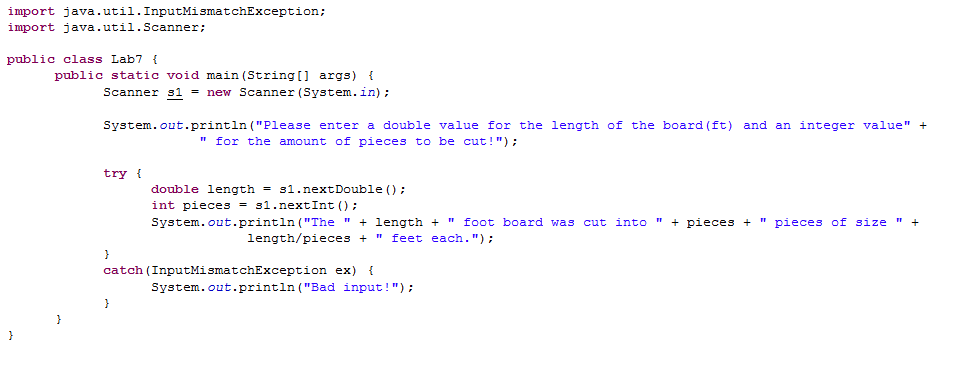


Here we have the basic solution. There is no need for any data fields here. We’ll simply create a scanner object with the System.in as the input. This will allow us to gather input from the command line. We then prompt the user to enter the length of the board and the amount of pieces to be cut. We accept these inputs and perform the calculation while printing back to the user.

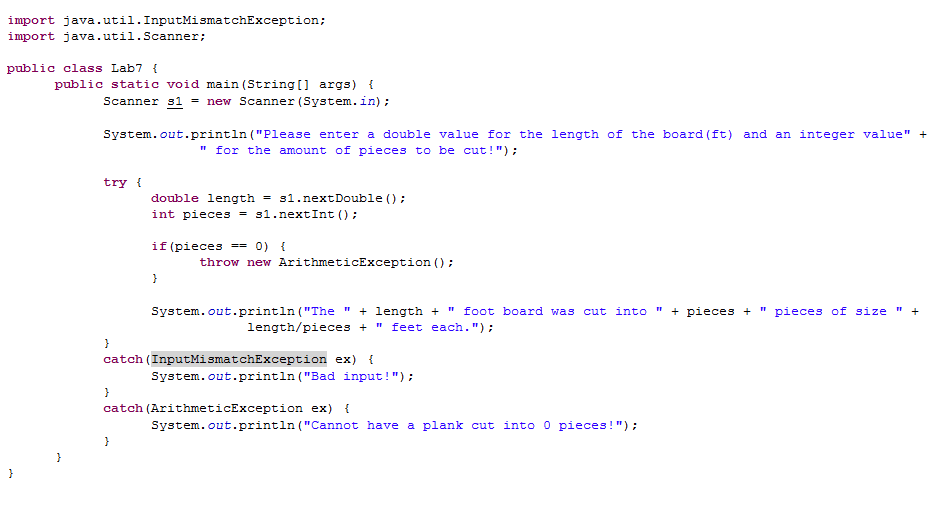
As we discussed last week, things can be troublesome whenever we depend on input from the user. He/she can really mess things up by not following the rules. What are some rules we need to follow here? From the lab, we know the amount of pieces can only be an integer. Let’s take care of that issue first.



This takes care of the pieces being a non-integer issue; however, it also takes care of all other input mismatch issues as well. For instance, if we input non-numerical values, they’ll be caught by this exception. So, let’s refine the error message before continuing.

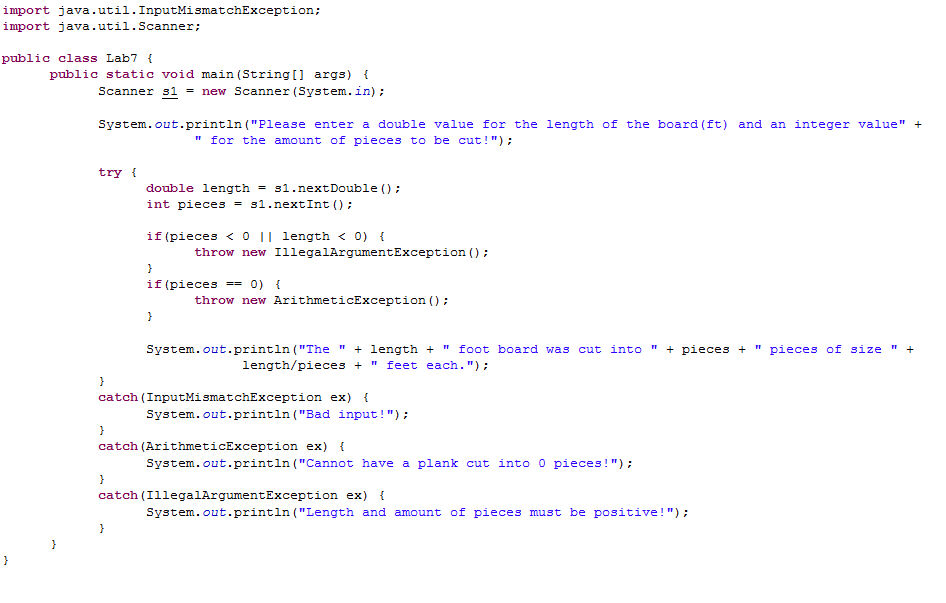


Now let’s take care of the issue of division-by-zero. Remember, as stated in the hints, we’ll want to manually create and throw an exception this time.

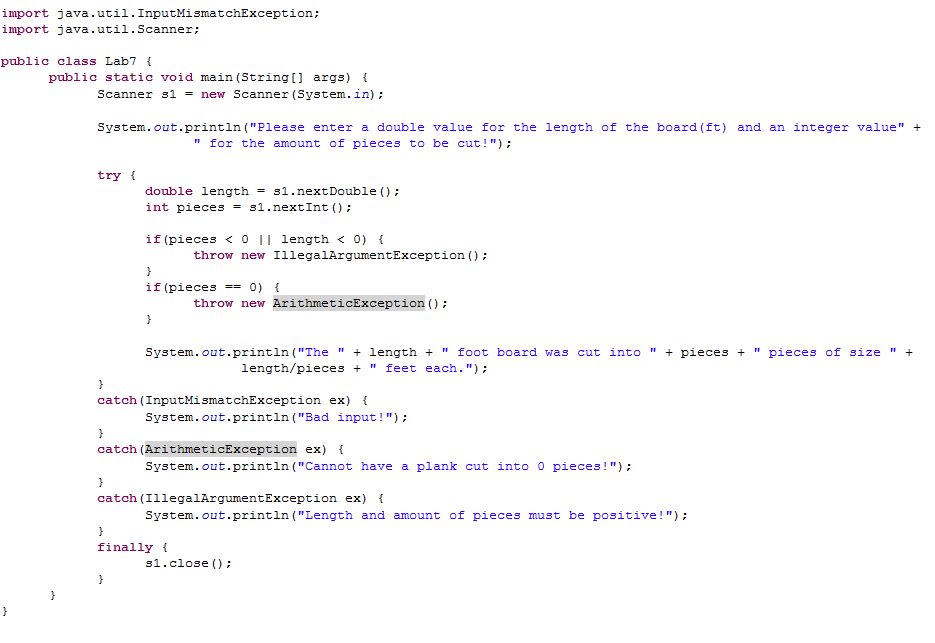


Note here we use the if-statement to decide whether or not to create and throw this exception. We handle it with another error message.

Now, let’s move onto negative inputs. There is no such thing as a plank with a negative length or a negative amount of pieces, so let’s use an IllegalArgumentException to restrict that range to positive values only.



We are almost done now. The only issue is that our scanner object, s1, is complaining that it has not been closed. Since we’ll want to close this in every situation, where would be the best spot?



That’s right, right there in the finally block. We know this code will be executed every time, and we always want to make sure that the scanner is closed when we are done with it.