2/25/2020

**Lab 5 Report - Group 10 - Spencer Manyet and Luke Cheng**

The objective of this Lab was to examine C++ templates and exception handling. In this lab, we built upon our understanding of arrays and implementing classes (using the show class we built in the previous class), to build a bookshelf. In addition, we learned to use C++ templates to build generic classes and learned to use exception handling.

At first, we built a base class which modeled a shelf of games, using Shows from our previous lab as “board games”. We tested this base class without testing the “extreme” cases. Afterwards, we built exception handling to catch the cases in which users would add a bookshelf to a full shelf or remove from an empty shelf. This would throw an error and prompt the user to try again.

This type of simple exception handling is important in CS careers because not all boundary cases may be handled simply by the code. Sometimes it is easier or better to handle the exceptions by doing exception handling. The OOP method of exception handling allows us to create classes with different names for different types of exceptions. This will be stored for the programmer, and then potentially some output will display to the user about the error.

At a simple level, exception handling is important because we don’t want our program to break for exceptions. We want the program to be able to continue running and the user to be able to go about their day. At an industry level, exceptions are important because code is written by different programmers and different times. This coordination leads to bugs which need to be dealt with. Instead of searching through the whole code to be debugged, errors that are handled can point to what actually occurred. This will save programmers a lot of time and hassle. It’ll make fixing the error much easier as well. For the user, at an extreme level for the user, if errors aren’t handled correctly, the user may lose all of their work or data. This would obviously be troublesome for any type of software. Clearly, exception handling is important.

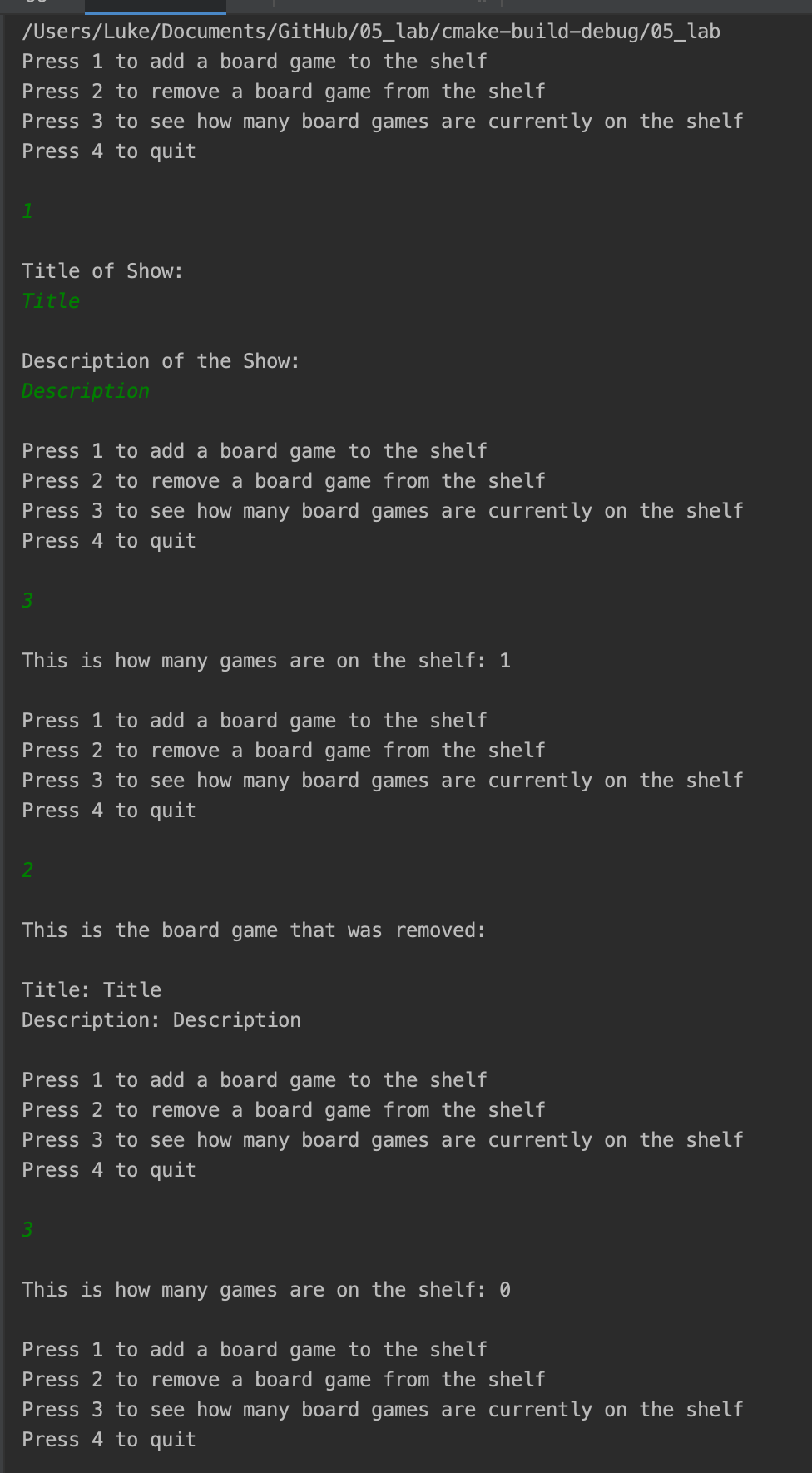
**Spencer writes about the importance of templates here.**

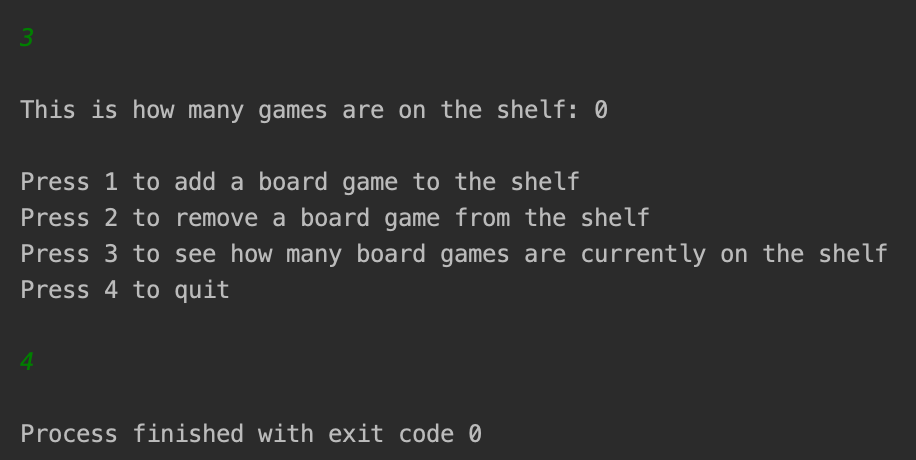
**Task 1:**

The add and remove methods were implemented very similarly to a stack, where add and remove were done exactly like a push and a pop for a stack.

**Task 2:**

Output of test:

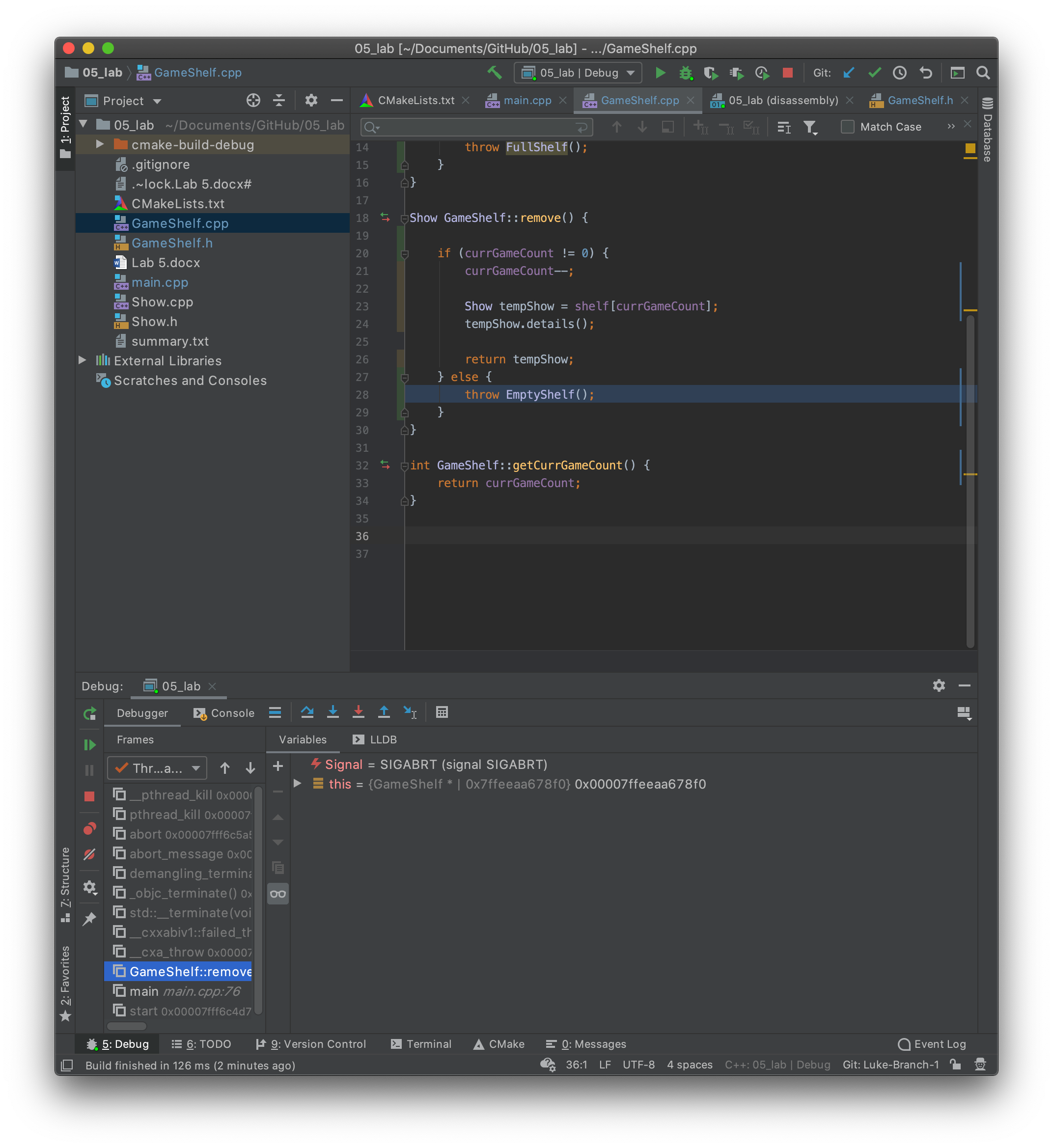




**Task 3:**

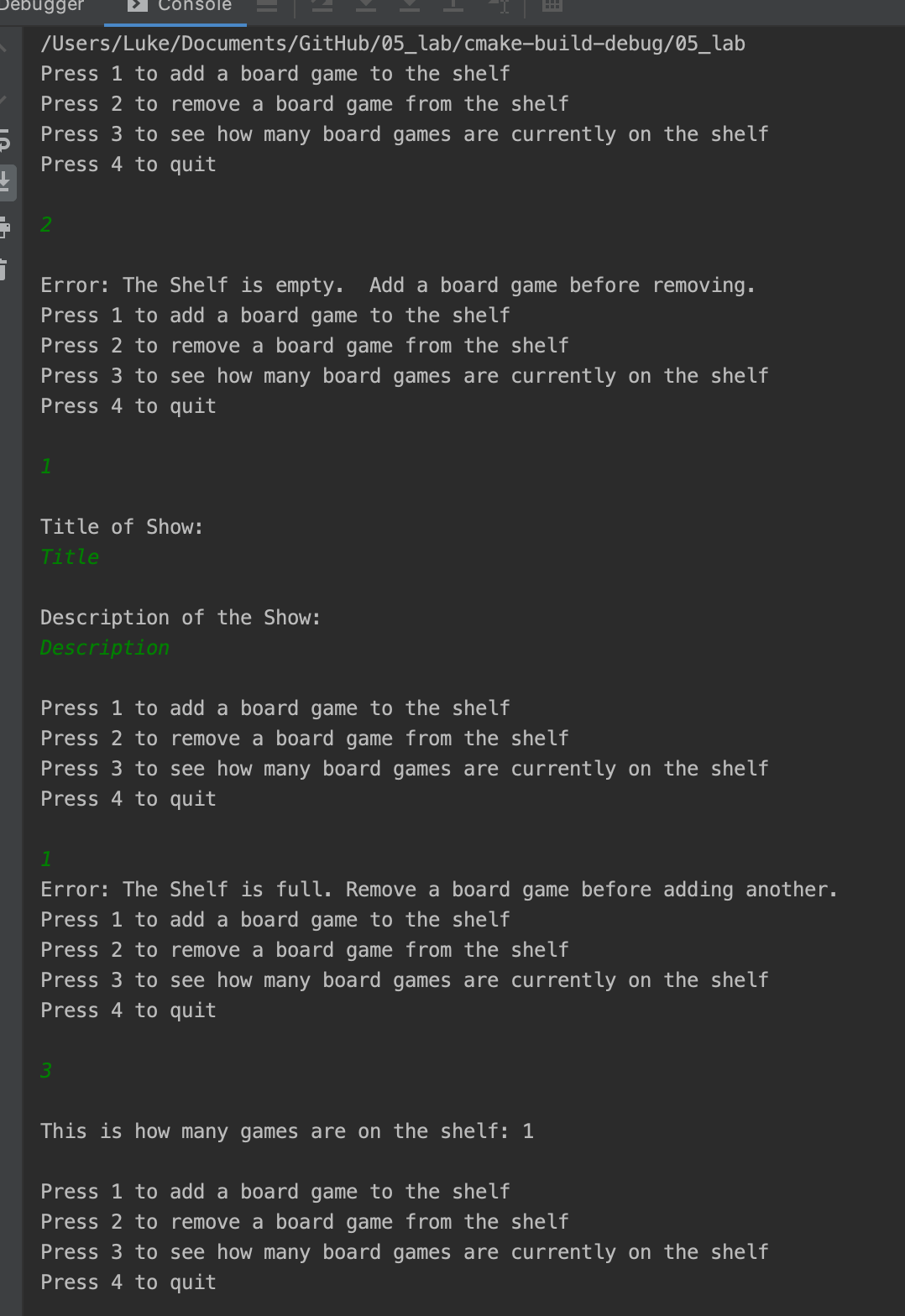
The first way exceptions were made was to stop the program and throw either EmptyShelf() when removing a board game from an empty shelf or FullShelf() when adding a board game to a full shelf.

This is what happened in the IDE. There was an error thrown and the line for this code was shown.



For the second part of this test, we included try catch blocks so that any errors would be caught and the program would continue. This makes the program infinitely better because the user can now continue using the software even with an error. In real life, this would be like a user confronting a 404 error or some other kind of error. You would want the error to be trapped and logged but you would want the user to be able to continue using the website. Usually it’s done in a way where the catch block would be to present some kind of 404 error screen. Then the user can continue using other parts of the website while they wait for that other piece of the website to be fixed. The error is logged and the programming team (or an automatic QA tester) will now decide if it needs fixing or if it’s just a user error.

This was our output:



When comparing handling errors within the calling function or trapping with a class, the principles of OOP come into play again. If we write a general purpose function with exception handling, we would need to think of all of the cases in which an error could be thrown using the function, instead of just the errors that could be thrown given the context of the code we are in. In addition, error classes can be reused with multiple functions to save us a lot of time. This cannot be the case with normal functions. Within try catch fields, we can also stack potential exception handling on top of each other in a specific way starting from more specific to broad errors. Again, this could be done with a function but would be tedious as we would need to think of every single case without referencing specific classes or exceptions. In addition, we can propogate error information back up the chain in nested try catch blocks. Trapping errors within a class will allow also the error to be handled gracefully with error messages. Lastly, trapping errors within a class allows programmers to build in functionality to find where the error happened. This may be the most important.

Overall, trapping errors with a class allows more reusability and functionality.

**Task 4:**

The use of a template for the EntertainmentCollection class gave it substantial added flexibility. The template was able to use any valid datatype to store on the shelf. However templates also create added complexity to the code and reduce readability.

