Names: Madison Betz and Kaiden Pollesch

Date: 3/11/2025

Assignment Title: Lab 6 Boundary Value Analysis - The Tax Code

Repository Link: https://github.com/msoe-SWE2721/swe2721-lab6-taxcode-121_betz_pollesch_lab6_2025.git

Introduction

What are you trying to accomplish with this lab?

In this lab, we are evaluating existing code on a simplified tax filing system using boundary value analysis and equivalence class testing. One of the partners will write the implementation and the other will work on the testing so that the final product is a functional tax calculator. We also are going to create automated unit tests which will reveal how our code is working and use tools to measure code coverage.

Input Domain Analysis / Boundary Value Analysis

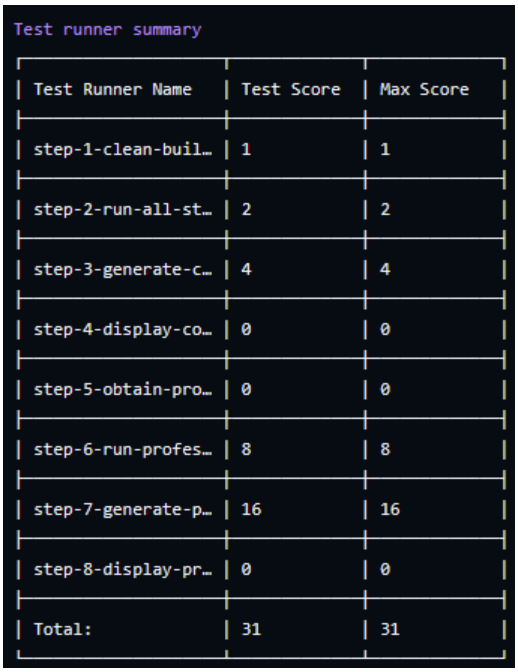
Input Domain Table

Method Name	Parameters	Internal State	Return Type	Return Values	Exceptions Thrown	Characteristic ID	Characteristic Description
getFilingStatus		FilingStatus	FilingStatus	Filing status		C1	Returns the taxpayer's filing status
getName			String	Name of the taxpayer		C2	Returns the name of the taxpayer
getSpouseName			String	Name of the spouse		C3	Returns the spouse's name or null if not applicable
getAge			Integer	Age of the filer		C4	Returns the age of the taxpayer
getSpouseAge			Integer	Age of the spouse		C5	Returns spouse's age or 0 if no spouse
setAdjustedGrossIncome	adjustedGrossIncome:double		void		TaxFilingException	C6	Sets the adjusted gross income, throws exception if negative
getAdjustedGrossIncome			Double	Gross income		C7	Returns the adjusted gross income
determineFilingNeed			Boolean	True if tax return is required, false if not		C8	Determines if a tax return is required
obtainStandardDeduction			Double	Standard deduction		C9	Returns the standard deduction based on status and age
obtainTaxableIncome			Double	Taxable income		C10	Returns taxable income
getTaxDue			Double	Tax due		C11	Returns the calculated tax due
getNetTaxRate			Double	Net tax rate		C12	Returns the tax due divided by gross income as a percentage

Test Coverage

Code Coverage Summary

This test we covered all the possibilities that we could think of as well as checking over the professor tests, as we passed all those tests. Since our code passed all of our tests and the teacher tests, we believe that we have covered as much as we could.



Test runner summary

Test Runner Name	Test Score	Max Score
step-1-clean-buil…	1	1
step-2-run-all-st…	2	2
step-3-generate-c…	4	4
step-4-display-co…	0	0
step-5-obtain-pro…	0	0
step-6-run-profes…	8	8
step-7-generate-p…	16	16
step-8-display-pr…	0	0
Total:	31	31

Defects

Bug	Description	Location
1	One bug that we had was we were checking the age of each single spouse before checking them at the same time, so they would always get into the first check, making them have the wrong adjustedGrossIncome	TaxCalculator2023:131
2	In both of the constructors in TaxCalculator2023 the paramaters were not validated so it didnt thro needed errors	TaxCalculator2023:30-51
3	Tax calculator didnt round to two decimal places, it kept going	TaxCalculator2023:197&208

Discussion

How did you go about designing the tests? Did you use a spreadsheet or other tool to help you come up with values and/or automate the process?
We went about designing the tests by using the input domain analysis to figure out what values needed to be tested for this. We also used tables that were in the lab handouts for different tax brackets and values that would need to be tested. Using these items made it much faster to write tests as the values were laid out for the most part.

How did data providers help to structure your tests?
Data providers helped to structure our tests because they had a set location for where the information passed into the tests would go. This allowed the testing file to be more structured and organized. The data providers set up what values would be passed into each test, which also helped show side-by-side what range of values had been tested.

How did you go about implementing your code? Did you use techniques like a table lookup, a map, or some other internal mechanism to simplify your implementation?
We went about implementing our code by using the tables provided in the lab handout and using the Javadocs from the interface given to us. These items helped simplify implementation of the code by laying out how different methods behaved and the expected return values. Using the input analysis domain also helped with knowing the expected behaviors of the methods.

How easy was your code to debug when the tester discovered a defect?
It was relatively easy to debug the code when the tester discovered a defect because the methods of the code are relatively specific and contain relatively simple code. What also really helped was having specific test cases, so it was easier to see exactly how the code failed.

Things Gone Right / Things Gone Wrong

Things that went right in this lab were creating the input domain analysis and implementing the code. Once we were able to write the code and tests, it went well and was simple enough to do because of the tables given to us in the lab handout, along with a provided interface and UML diagram.
Things that went wrong in this lab were getting the code set up initially. We had problems with the GitHub teams and not being placed in the right one, which caused us to be delayed in starting to code. For some reason, GitHub kept placing one of our team members in a different group regardless of what they clicked.

Conclusion

What have you learned from this experience?
From this experience, we learned more about using boundary value analysis and equivalence class techniques in testing. We were able to look for more edge cases along with expected and unexpected outputs by these testing methods. Boundary value analysis allowed us to test the bounds of code to make sure that it works on the limits of values and ranges for inputs/outputs.

How has this lab experience changed your attitude toward testing and when should you work on testing your projects?
This lab has changed our attitude toward testing and on how testing should be created to test all values, both expected and unexpected. Using the boundary value analysis allowed us to make sure that we tested values nearing the outside our bounds for edge cases. The code coverage summary showed what areas of the code were well tested and what areas might need more test cases to fully cover them and make sure the untested code was implemented correctly.