Refactoring Of Minnesota Income Tax Calculator Application

Overall Report

Κοσμάς Αποστολίδης 4259 cs04259@uoi.gr

Πέτρος Καραμπάς 2987 cs02987@uoi.gr

Table of Contents

Introduction 2

Build And Run 3

Refactored Design 3

UseCases 3

Architecture 8

DetailedDesign 8

Classes Responsibilities and Collaborations (CRC CARDS) 15

Introduction

The goal of this project is to refactor the legacy code to be more efficient and extensible.We are going to detail Use Cases, describe the new GUI redesign and go into some detail about the refactors that were applied to the code.

If you encounter any problems running this project please contact us.

cs02987@uoi.gr

cs04259@uoi.gr

Build and Run

For the build process we took advantage of the Maven build tool to handle the project dependencies and for easier project management. In order to run this project using Eclipse IDE you need to import it as a **Maven** project and have **Java 11**. You might need to Update the project by right clicking the project folder inside Eclipse and clicking Maven, then Update Project.

If you encounter problems running this project please contact us.

To run this project follow these steps:

1. Import the project as a Maven project.
2. Make sure you have Java 11 installed.
3. Run Bootloader.java as a Java Application

Refactored Design

Use Cases

The following use cases have been written based on our GUI implementation. Some of them don't have alternative flows, because our GUI implementation doesn't let the user raise an alternative flow.

Delete taxpayer

Use case ID	UC1
Actors	Ordinary User
Preconditions	The application is up and running.

Main flow of events	1. The use case starts when the user wants to delete an existing taxpayer. 2. The user selects an existing taxpayer's AFM from the GUI. 3. The user confirms the selection.
Postconditions	The application deletes the specified taxpayer.

Save Data

Use case ID	UC2
Actors	Ordinary User
Preconditions	The application is up and running.
Main flow of events	1. The use case starts when the user wants to save the taxpayer’s data into file. 2. The user specifies the format of the file(.txt or .xml). 3. The user confirms the selection.
Postconditions	The application saves the taxpayer’s information to a <AFM>_LOG.txt or <AFM>_LOG.xml file with the taxpayer’s data.

View Chart Report

Use case ID	UC3
Actors	Ordinary User
Preconditions	The application is up and running.

Main flow of events	1. The use case starts when the user wants to visualize the taxpayer’s data. 2. The user presses the View Chart Report button on the GUI.
Postconditions	The application creates a bar chart with the basic tax, tax increase/decrease and total tax in the x-axis and a pie chart with the values of the taxpayer’s receipts based on the different receipt kinds.

Delete Receipt

Use case ID	UC4
Actors	Ordinary User
Preconditions	The application is up and running.
Main flow of events	1. The use case starts when the user wants to delete an existing receipt. 2. The user selects from the GUI the receiptID that wants to delete. 3. The user confirms the deletion of the receipt.
Postconditions	The application deletes the specified receipt with the selected receiptID and then reloads the contents of the <AFM>_INFO file.

Add Receipt

Use case ID	UC5
Actors	Ordinary User
Preconditions	The application is up and running.
Main flow of events	1. The use case starts when the user wants to add a new receipt. 2.The user specifies the receipt id, date, kind, amount, company, country, city, street and street number. 3. The user confirms the new receipt.
Alternative flow 1	1. The alternative flow starts after step 2. 2. The application indicates that the receipt id already exists. 3. The user changes the receipt id with a non existing one. 4.The use case continues from step 3 of the main flow
Postconditions	The application adds the new receipt and then it reloads the contents of the <AFM>_INFO file.

Select Taxpayer

Use case ID	UC6
Actors	Ordinary User
Preconditions	The application is up and running.
Main flow of events	1. The use case starts when the user wants to select an existing taxpayer from the GUI.

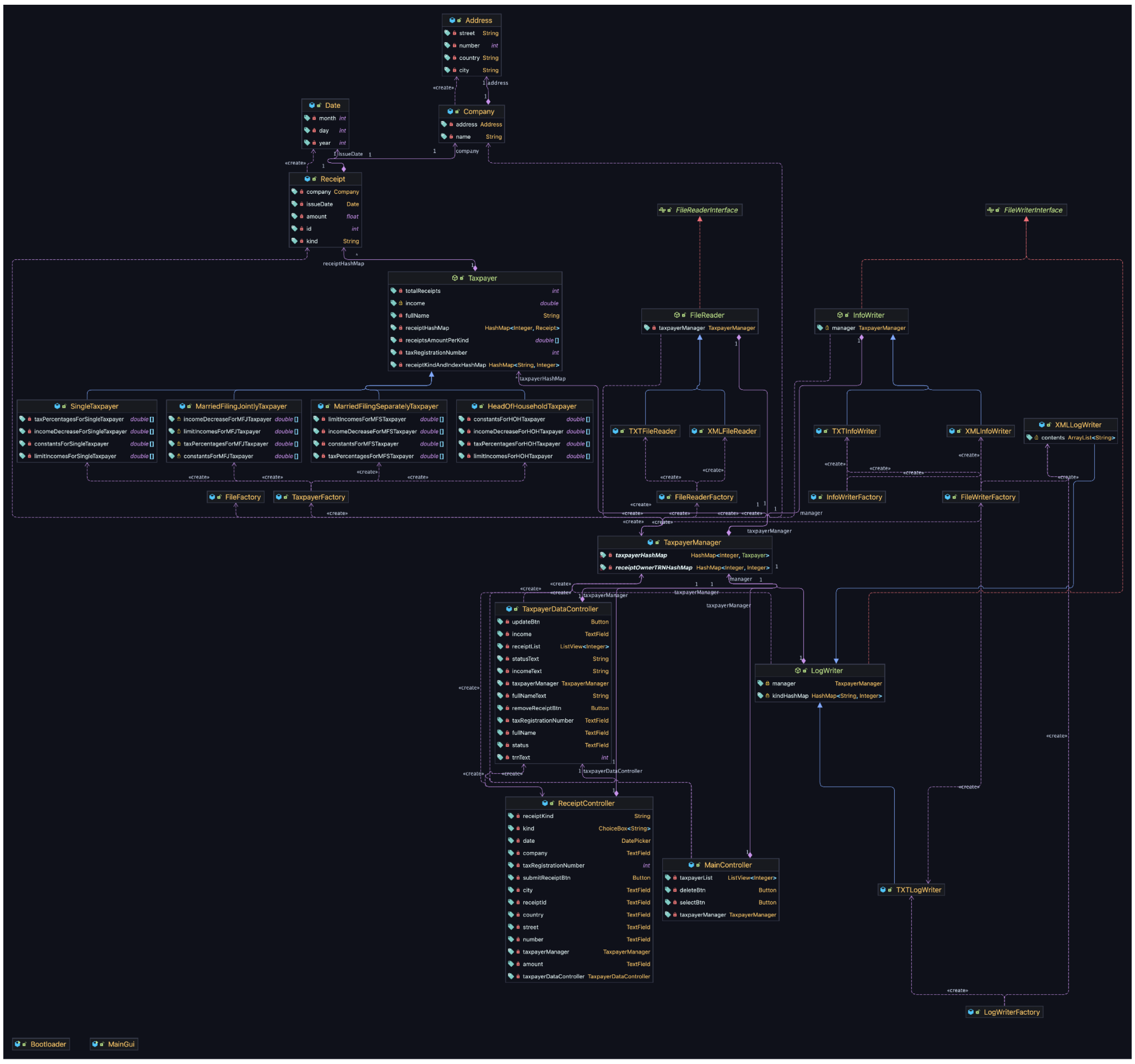
	2. The user selects the Tax Registration Number that is loaded in the GUI . 3. The user confirms the selection.
Postconditions	The application loads the information of the specified tax registration number.

Load Taxpayer

Use case ID	UC7
Actors	Ordinary User
Preconditions	The application is up and running.
Main flow of events	1. The use case starts when the user wants to load a taxpayer. 2. The user specifies the Tax Registration Number. 3. The user confirms the selection.
Postconditions	The application shows the specified tax registration number to the user.

Architecture

Detailed Design



Address
street String
number int
country String
city String
Company
address Address
name String
Date
month int
day int
year int
Receipt
company Company
issueDate Date
amount float
id int
kind String
Taxpayer
totalReceipts int
income double
fullName String
receiptHashMap HashMap<Integer, Receipt>
receiptsAmountPerKind double []
taxRegistrationNumber int
receiptKindAndIndexHashMap HashMap<String, Integer>
SingleTaxpayer
taxPercentagesForSingleTaxpayer double []
incomeDecreaseForSingleTaxpayer double []
constantsForSingleTaxpayer double []
limitIncomesForSingleTaxpayer double []
MarriedFilingJointlyTaxpayer
incomeDecreaseForMFJTaxpayer double []
limitIncomesForMFJTaxpayer double []
taxPercentagesForMFJTaxpayer double []
constantsForMFJTaxpayer double []
MarriedFilingSeparatelyTaxpayer
limitIncomesForMFSTaxpayer double []
incomeDecreaseForMFSTaxpayer double []
constantsForMFSTaxpayer double []
taxPercentagesForMFSTaxpayer double []
HeadOfHouseholdTaxpayer
constantsForHOHTaxpayer double []
incomeDecreaseForHOHTaxpayer double []
taxPercentagesForHOHTaxpayer double []
limitIncomesForHOHTaxpayer double []
FileFactory
TaxpayerFactory
FileReaderInterface
FileWriterInterface
FileReader
taxpayerManager TaxpayerManager
TXTFileReader
XMLFileReader
FileReaderFactory
InfoWriter
manager TaxpayerManager
TXTInfoWriter
XMLInfoWriter
XMLLogWriter
contents ArrayList<String>
InfoWriterFactory
FileWriterFactory
TaxpayerManager
taxpayerHashMap HashMap<Integer, Taxpayer>
receiptOwnerTRNHashMap HashMap<Integer, Integer>
TaxpayerDataController
updateBtn Button
income TextField
receiptList ListView<Integer>
statusText String
incomeText String
taxpayerManager TaxpayerManager
fullNameText String
removeReceiptBtn Button
taxRegistrationNumber TextField
fullName TextField
status TextField
trnText int
ReceiptController
receiptKind String
kind ChoiceBox<String>
date DatePicker
company TextField
taxRegistrationNumber int
submitReceiptBtn Button
city TextField
receiptId TextField
country TextField
street TextField
number TextField
taxpayerManager TaxpayerManager
amount TextField
taxpayerDataController TaxpayerDataController
MainController
taxpayerList ListView<Integer>
deleteBtn Button
selectBtn Button
taxpayerManager TaxpayerManager
LogWriter
manager TaxpayerManager
kindHashMap HashMap<String, Integer>
TXTLogWriter
LogWriterFactory
Bootloader
MainGui
«create»
address
company
issueDate
receiptHashMap
taxpayerHashMap
taxpayerManager
manager
taxpayerDataController

GUI Redesign

For the GUI we switched to JavaFX 19.

We redesigned the GUI to be more intuitive and easy to navigate and understand. In the main window the user can now load a file by browsing By simply clicking the load button and choosing one of the two options (TXT or XML) the user can load the file of his choosing (ΑΦΜ_INFO.txt/xml). The files however need to be inside the project's folder. That is because the legacy code was not extensible enough to add that feature so we decided to keep it as is.

The loaded taxpayers will be loaded in a list on the left side of the window. The user can delete any one of the taxpayers by simply clicking on one of them and then clicking the delete button.

The user no longer selects by typing in the tax registration number to access taxpayer information but by simply clicking on the tax registration number and then clicking the View button.

In the taxpayer data scene we can on the left side we can view the personal details of the selected taxpayer and on the right side we have a list of the taxpayer's receipts.

Again here the point is simplicity. In order to delete a receipt the user no longer needs to type the receipt id but simply selecting the id from the list and clicking the **Remove Receipt** button. That way we can also prevent the user from selecting a receipt id that does not exist.

We have also introduced a way to edit some of the taxpayer's information such as the **FullName** and the **Income.** The user can edit these fields right in the window and update them by clicking the **Update**

Data Button. The user can also save a LOG file or view a chart report by clicking the **Save Log File** Button in either **.txt** or **.xml** format or the View Chart Report Button respectively.

Lastly the User can add a new receipt by clicking the **Add Receipt** button and a new window to do so.

The **Add Receipt** window was also improved and simplified. The core features are still the same but the main changes are two. We have added a date picked so the user doesn't have to input a date but a calendar shows up and the user can simply choose the desired date. This way we eliminated the possibility of the user entering a wrong date format that is not supported by the program. Another improvement is choosing the receipt type by a drop down menu. That way it is not possible to input a wrong receipt type. Lastly we made it so the user has to fill all the fields of the receipt form in order to submit it otherwise the window can be closed

Below we discuss the different problems we faced during the refactor of classes and methods.

incometaxcalculator.data.management package

- **Company class:** This class had an unnecessary code with no usage, so it was removed.
- **Taxpayer class:** This class had some complex conditional logic in the form of chained if-else statements at the following methods: *addReceipt()*, *removeReceipt()*, *getVariationTaxOnReceipts()*. We switched these chained statements with **for loops** with the purpose of simplifying the conditional logic of these methods.

- **Subclasses of the Taxpayer class:** Those classes had some duplicate code inside the **calculateBasicTax()** method. So, for that problem we used code parameterization. Firstly, we placed the different constants for the different taxpayers types at 4 arrays(one for all the different constants). The result was, having the same method in every subclass, which could be pulled up to the base Taxpayer class. The subclasses, then, can be used to initialize the values of those arrays, inside of their respective constructors, so that they can do the calculation of the basic tax.
- **TaxpayerManager class:** This was a **Large Class** with many responsibilities.
    1. **createTaxpayer()** method had some conditional logic with the form of if-else statements, that was used for the creation of the different types of Taxpayer objects. We removed this conditional logic with the creation of a simple parameterized Taxpayer factory. The factory creates a specific Taxpayer object based on the input status given to the factory.
    2. **updateFiles()** method had some conditional logic that was used for the creation of the different types of FileWriter objects. We removed this conditional logic by creating a simple parameterized FileWriterFactory. This factory creates a specific FileWriterInterface object based on the input String format given to the factory.
    3. **saveLogFile()** method had some conditional logic that created different types of FileWriter objects, e.g TXTLogWriter and XMLLogWriter. We moved that logic to a simple parameterized FileWriterFactory.

4. **loadTaxpayer()** method had some common parts that were simplified after the creation of **FileReaderFactory** for the creation of the different FileReader objects based on the format of the file.

incometaxcalculator.data.io package:

- **TXTFileReader, XMLFileReader classes**: These two classes had some **duplicate code**. Each one of these classes had two methods with a similar body, checkForReceipt() and getValueOfField(). For this reason we extracted the parts of the code that are different in simple abstract methods (getReceiptID() and getLastValueFromLine()) that are implemented in the subclasses (TXTFileReader and XMLFileReader). In addition, we pulled up the identical body checkForReceipt() and getValueOfField() in the base FileReader class. Also, we simplified the body of the readReceipt() method by creating a method called readReceiptFromFile() that does the receipt reading from the given file.
- **FileWriter class**: This class was a **Middle Man** for the TaxpayerManager class. More specifically, it contained some methods that some other classes could call, instead of the basis TaxpayerManager methods. So, for that reason, the delegating methods were removed and instead of these methods, these classes, in the end, call directly the TaxpayerManager method and for practical reasons, the FileWriter class changed to an interface.
- **TXTInfoWriter and XMLInfoWriter classes:** These two classes had some **duplicate code**. Each one of them contains methods with similar code. The solution for that was, creating a base abstract class, **InfoWriter** , that implements the

FileWriterInterface. Then, the two subclasses extend the InfoWriter class, which contains abstract methods for the implementation of the similar steps. In other words,the **Template** design pattern was implemented for solving this problem.

- **TXTLogWriter and XMLLogWriter classes:** Here we faced the same problem,the problem of the **duplicate code** ,that also appeared in these two classes. The creation of a base abstract **LogWriter** class was created that implements the **FileWriterInterface**. The LogWriter class contains abstract methods for the implementation of the similar steps that TXTLogWriter and XMLLogWriter classes had before the refactoring.

Classes Responsibilities and Collaborations (CRC CARDS)

Class Name: TaxpayerManager	
Responsibilities	**Collaborations**
• Adds a non existing receipt • Removes an existing receipt • Loads a taxpayer • Removes taxpayer	• TaxpayerFactory • FileWriterFactory • FileReaderFactory • FileFactory • Taxpayer • WrongTaxRegistration NumberException • ReceiptAlreadyExists Exception • WrongTaxpayerStatus Exception • WrongReceiptIDException • Company • Receipt • MainController

Class Name: Taxpayer	
Responsibilities	**Collaborations**
● This class is responsible for calculating the basic and total tax of a given taxpayer	● Receipt ● TaxpayerManager

Class Name: MarriedFilingSeparatelyTaxpayer	
Responsibilities	**Collaborations**
● Calculates the basic tax for a married filing separately taxpayer	

Class Name: MarriedFilingJointlyTaxpayer	
Responsibilities	**Collaborations**

● Calculates the basic tax for a married filing jointly taxpayer	

Class Name: HeadOfHouseholdTaxpayer	
Responsibilities	**Collaborations**
● Calculates the basic tax for a head of household taxpayer	

Class Name: SingleTaxpayer	
Responsibilities	**Collaborations**
● Calculates the basic tax for a single taxpayer	

Class Name: TaxpayerFactory	
Responsibilities	**Collaborations**
● Creates the different kinds of taxpayer items	● MarriedFiling SeparatelyTaxpayer ● MarriedFiling JointlyTaxpayer ● HeadOfHouseholdTaxpayer ● SingleTaxpayer

Class Name: FileFactory	
Responsibilities	**Collaborations**
● Creates the different types of FILE items	

Class Name: FileReaderFactory	
Responsibilities	**Collaborations**
● Creates the different types of FileReader items	● FileReader ● TXTFileReader ● XMLFileReader ● WrongFileFormatException

Class Name: FileWriterFactory	
Responsibilities	**Collaborations**
● Creates the different types of FileWriterInterface items	● TXTInfoWriter ● XMLInfoWriter ● TXTLogWriter ● XMLLogWriter ● WrongFileFormatException

Class Name: TXTInfoWriter	
Responsibilities	**Collaborations**
● Writes to a <AFM>_INFO.txt the taxpayer info(Name, AFM, Status, Income, Receipts)	●

Class Name: XMLInfoWriter	
Responsibilities	**Collaborations**

● Writes to a <AFM>_INFO.xml the taxpayer info(Name, AFM, Status, Income, Receipts)	

Class Name: TXTLogWriter	
Responsibilities	**Collaborations**
● Writes to a <AFM>_LOG.txt the Name, AFM, Income, Basic Tax, TaxIncrease/Decrease, Total Tax, TotalReceiptsGathered, and all receipts amount per kind.	

Class Name: XMLLogWriter	
Responsibilities	**Collaborations**
● Writes to a <AFM>_LOG.xml the Name, AFM, Income, Basic Tax, TaxIncrease/Decrease, Total Tax, TotalReceiptsGathered, and all receipts amount per kind.	

Class Name: LogWriter	
Responsibilities	**Collaborations**
● Generates the log file with the specified tax registration number	● TaxpayerDataController

Class Name: InfoWriter	
Responsibilities	**Collaborations**

● Generates the info file with the specified tax registration number	● TaxpayerDataController

Class Name: TXTFileReader	
Responsibilities	**Collaborations**
● Reads the contents from a .txt file	

Class Name: XMLFileReader	
Responsibilities	**Collaborations**
● Reads the contents from the .xml file	

Class Name: FileReader	
Responsibilities	**Collaborations**

● Reads the contents from the file based on format (.txt or .xml)	● TaxpayerManager

Class Name: Date	
Responsibilities	**Collaborations**
● Represents the date(day/month/year)	

Class Name: Receipt	
Responsibilities	**Collaborations**
● Represents the receipt(id, issueDate, amount, kind, company)	● Date ● Company

Class Name: Company	
Responsibilities	**Collaborations**

• Represents the company(name, country, city, street, number)	• Address

Class Name: Address	
Responsibilities	**Collaborations**
• Represents the address(country, city, street, number)	• Company