MYE004 -SOFTWARE DEVELOPMENT II

REFACTORING OF MINNESOTA INCOME TAX CALCULATION

OVERALL REPORT

Kondylia Vergou 4325

DECEMBER 2022

TABLE OF CONTENTS

INTRODUCTION

This project aims to reengineer the legacy Java application of the Minnesota Income Tax Calculator.

A quick summary of the application:
The application serves for the income tax calculation of the Minnesota state citizens.
The tax calculation is based on a complex algorithm provided by the Minnesota state and the necessary data for it is loaded from txt or xml files. Each file contains information about a citizen, such as marital status, income, and an amount of that they have spent, as witnessed by a set of receipts declared along with the income.
The application further produces graphical representations of the data in terms of bar and pie charts.
Finally, the application produces respective output reports in txt or xml.

The objectives of the project can be categorized as:

1. **Refactoring:**	Locate code smells in the project and refactor them.
2. Improvement:	Fix bugs that may exist in the legacy application. Provide jUnit Tests for important and use-cases methods. Replace the GUI with a new one that is easier to learn and use. Add new functionalities to the GUI.
3. Documentation:	Specify the use cases of the legacy application. Provide UML package and class diagrams. Report code changes that address the problems of the previous designs. Provide CRC cards for the classes of the refactored project.

REFACTORED DESIGN

USE CASES

UC01: LoadTaxpayer	
Use case ID	01
Actors	User
Preconditions	-
Main flow of events	1. The use case begins when the user clicks the button “Load Taxpayer” from the main menu. 2. The user chooses the info file of the taxpayer that they want to load from the file browser. 3. The system loads the taxpayer into the taxpayer database. 4. The system displays the taxpayer’s registration number in the main menu’s list of loaded taxpayers.
Alternative flow 1	If the file doesn’t exist, the system shows a warning.
Alternative flow 2	If the file’s name or contents are in incorrect format, then nothing is loaded.
Post conditions	The Taxpayer is loaded in the database.

UC02: SelectTaxpayer	
Use case ID	02
Actors	User
Preconditions	At least one Taxpayer is loaded in the system.
Main flow of events	1. The use case begins when the user selects a Taxpayer’s registration number from the list in the main menu. 2. The user clicks the button “Select Taxpayer” from the main menu. 3. The user is prompted to confirm their action. 4. The system displays a new window with information about the specific Taxpayer.
Alternative flow 1	If the user doesn’t select any of the loaded registration numbers, the button “Select Taxpayer” is not enabled.
Alternative flow 2	If the user cancels their action in step 3, there is no new window.
Post conditions	-

UC03: DeleteTaxpayer	
Use case ID	03
Actors	User
Preconditions	At least one Taxpayer is loaded in the system.
Main flow of events	1. The use case begins when the user selects a Taxpayer's registration number from the list in the main menu. 2. The user clicks the button "Delete Taxpayer" from the main menu. 3. The user is prompted to confirm their action. 4. The system removes the Taxpayer's registration number from the list in the main menu.
Alternative flow 1	If the user doesn't select any of the loaded registration numbers, the button "Delete Taxpayer" is not enabled.
Alternative flow 2	If the user cancels their action in step 3, nothing is displayed.
Post conditions	The Taxpayer is deleted from the database.

UC04: AddReceipt	
Use case ID	04
Actors	User
Preconditions	1. At least one Taxpayer is loaded in the system. 2. A Taxpayer is already selected, and their info window is opened.
Main flow of events	1. The use case begins when the user clicks the button "Add Receipt" from the Taxpayer's info window. 2. The system displays a form with fields for the new receipt. 3. The user fills out all the fields and clicks "Add Receipt" in the form window. 4. The system displays the receipt's id in the Taxpayer window.
Alternative flow 1	If the user doesn't fill out all the fields, there is a warning.
Alternative flow 2	If the user clicks "Cancel" in the form window in step 3, nothing is added.
Post conditions	The system adds the receipt in the Taxpayer's database and updates the corresponding info file.

UC05: DisplayReceipt	
Use case ID	05
Actors	User
Preconditions	At least one Taxpayer is loaded in the system and already selected.
Main flow of events	1. The use case begins when the user selects a receipt id from the list of receipts in the Taxpayer window. 2. The user clicks the button “Display Receipt” from the Taxpayer’s info window. 3. The user is prompted to confirm their action. 4. The system displays the receipt’s details in a new window.
Alternative flow 1	If the user doesn’t select any of the receipt ids, the button “Display Receipt” is not enabled.
Alternative flow 2	If the user cancels their action in step 3, nothing is displayed.
Post conditions	-

UC06: DeleteReceipt	
Use case ID	06
Actors	User
Preconditions	At least one Taxpayer is loaded in the system and already selected.
Main flow of events	1. The use case begins when the user selects a receipt id from the list of receipts in the Taxpayer window. 2. The user clicks the button “Delete Receipt” from the Taxpayer’s info window. 3. The user is prompted to confirm their action. 4. The system removes the receipt’s id from the Taxpayer window.
Alternative flow 1	If the user doesn’t select any of the receipt ids, the button “Delete Receipt” is not enabled.
Alternative flow 2	If the user cancels their action in step 3, nothing is deleted.
Post conditions	The system deletes the receipt id from the Taxpayer’s database and updates the corresponding info file.

UC07: GenerateCharts	
Use case ID	07
Actors	User
Preconditions	At least one Taxpayer is loaded in the system and already selected.
Main flow of events	1. The use case begins when the user clicks the button “Generate Charts” from the Taxpayer’s info window. 2. The system produces a bar chart with the tax analysis and a pie chart with the receipt amount analysis.
Post conditions	-

UC08: ExportLogFile	
Use case ID	08
Actors	User
Preconditions	At least one Taxpayer is loaded in the system and already selected.
Main flow of events	1. The use case begins when the user clicks the button “Export Log File” from the Taxpayer’s info window. 2. The system displays a prompt window with the file formats of the log file. 3. The user chooses one of the file formats. 4. The user clicks the button “Export Log File” in the prompt window. 5. The system produces a log file with the Taxpayers information.
Post conditions	-

Except for the UCO5 all the other use cases are from the legacy application.

Junit tests are provided for the back-end methods of the use cases.
However, UC02, UCO5 and UC07 are heavily GUI based and won’t be tested.

ARCHITECTURE

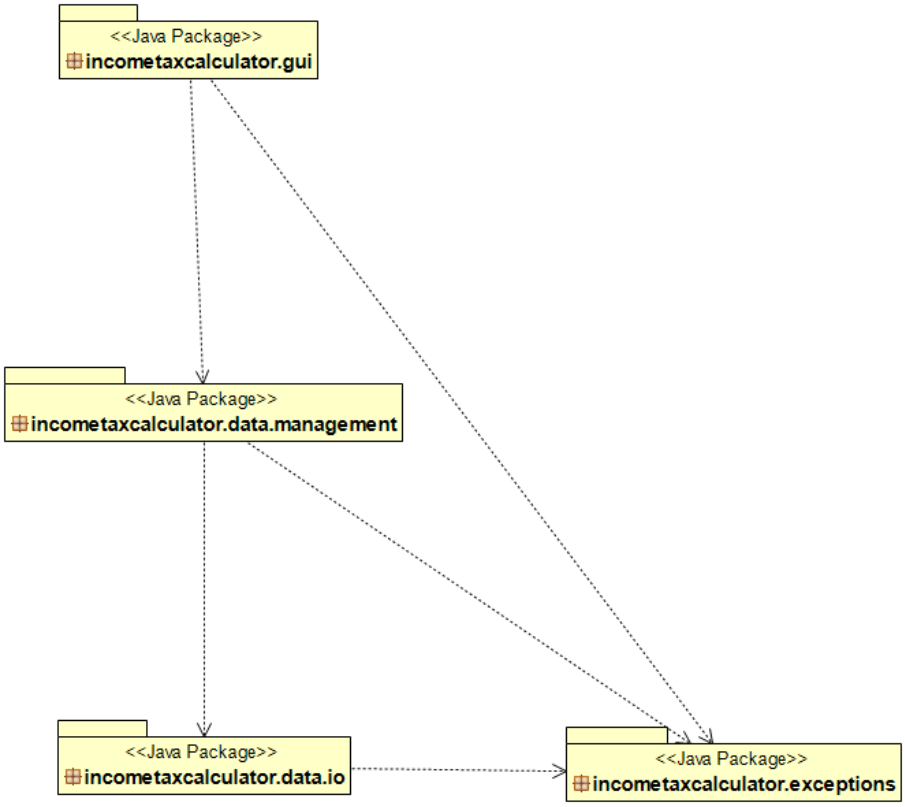

Figure 1: Overall UML Package Diagram

System packages	
incometaxcalculator.gui	Contains classes that implement the interface between the program and user
incometaxcalculator.data.management	Domain classes of the system
incometaxcalculator.data.io	Classes that process the input data of the app
incometaxcalculator.data.exceptions	Contains classes that handle exceptions

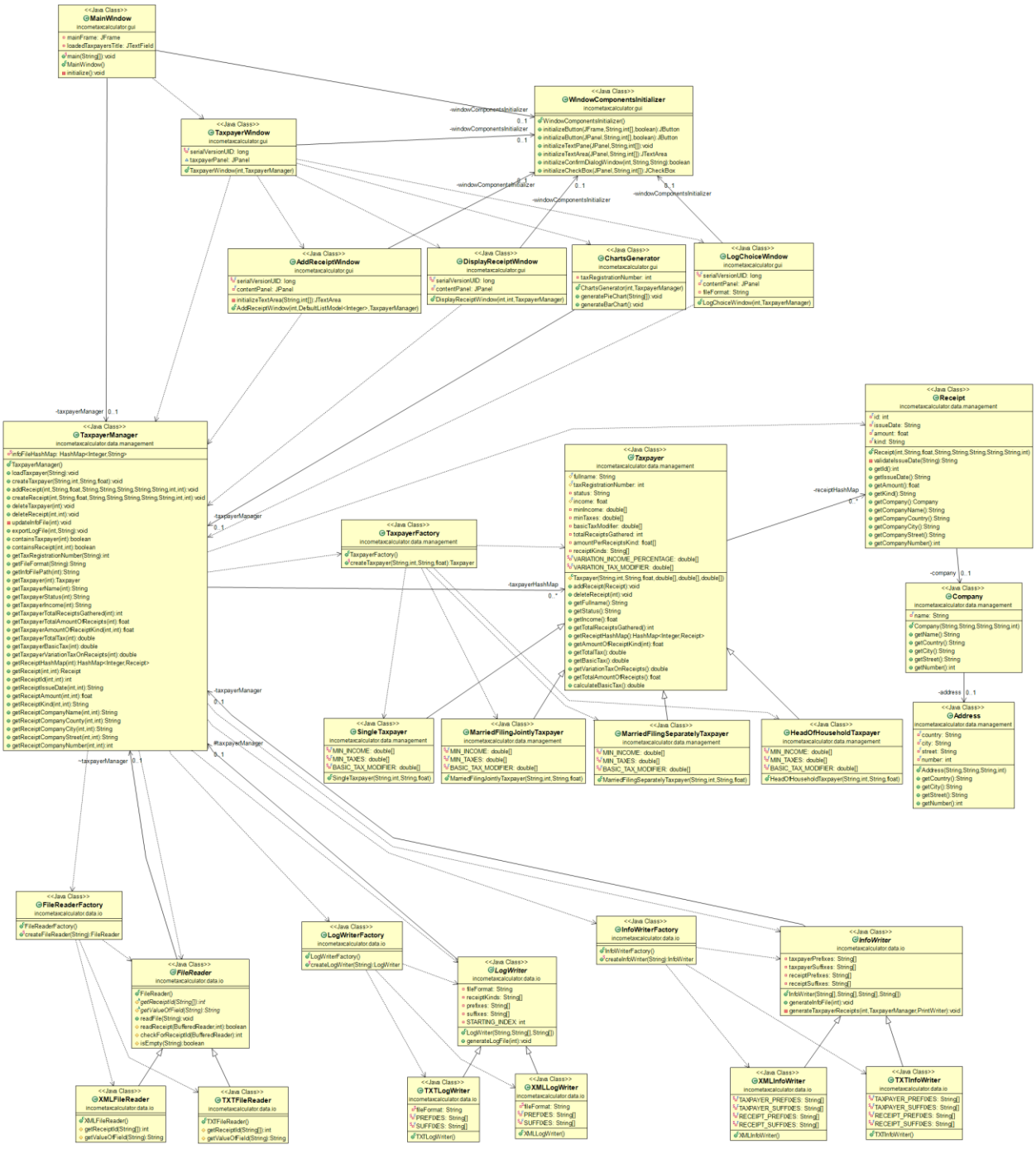

Figure 2: UML Diagram of the Whole Project (without the Exceptions classes)

DETAILED DESIGN

PACKAGE INCOMETAXCALCULATOR.DATA.MANAGEMENT

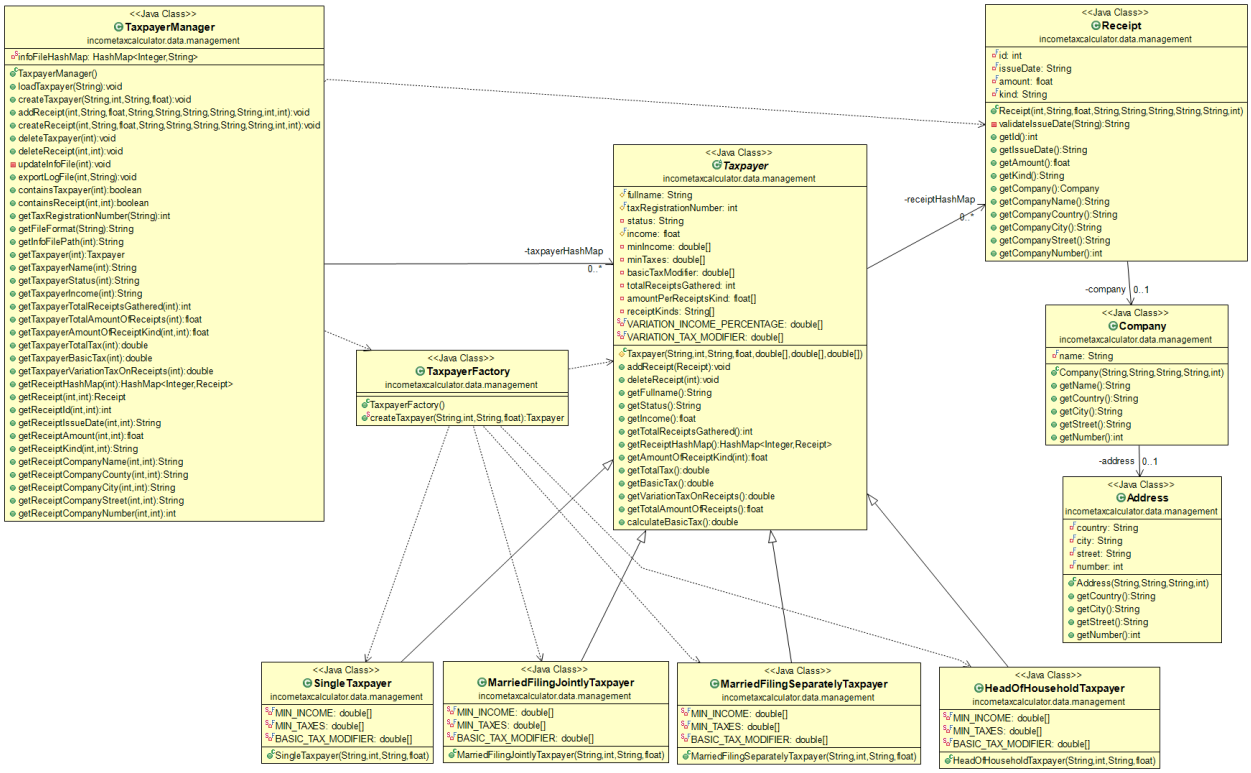

Figure 3: UML Class Diagram of the Package incometaxcalculator.data.management

The problem of the **Date** class was Unnecessary Complexity, therefore it was completely removed.

The **Company** and **Address** class contained getters that were dead code, so they were removed.

The **Receipt** class was modified to improve its design.

- The issue date is validated if it's in the correct format.
- The constructor is now responsible for the creation of the Company field. This responsibility was allocated from the TaxpayerManager to the Receipt class.
- Getters were added to implement the UC05 DisplayReceipt

In the **Taxpayer** class and its **subclasses** there is a Duplicate Code problem. The method calculateBasicTax is the same for every subclass except for some constant values.
To solve the problem, the following steps were taken:

- use array fields in the Taxpayer class for storing the different constants
- initialize the arrays in the subclasses
- change the method's body to use data structures

- transfer the implementation of the method in the Taxpayer Class and remove it from the subclasses

Furthermore, the removeReceipt method was renamed to deleteReceipt. In the following classes and the GUI, the 'remove' verb was replaced by 'delete' to use only One Name Per Concept.

Another problem in the Taxpayer class is the complex conditional logic in the addReceipt and deleteReceipt functions. The if-else was replaced by using a for loop and a receiptKinds array.
The conditional logic in the getVariationTaxOnReceipts method was also replaced by a for loop and arrays that hold the variation constants.

The **TaxpayerManager** class is a Large Class that has many responsibilities. There are 4 methods that take on the responsibility of creating the different kinds of objects that they need. The class can be simplified by delegating these actions to subordinate factory classes.

- createTaxpayer(): The conditional logic was moved to the **TaxpayerFactory** class
- loadTaxpayer(): The conditional logic was moved to the **FileReaderFactory** class
- updateInfoFile(): The conditional logic was moved to the **InfoWriterFactory** class
- exportLogFile(): The conditional logic was moved to the **LogWriterFactory** class

The methods updateFiles and saveLogFile were renamed to updateInfoFile and exportLogFile to use Intention Revealing Names.

In the **TaxpayerManager** class there was a very important bug. In the legacy application the HashMap receiptOwnerTRN was used to map the receipt ids to their respective owners. However, by using the receipt id as the HashMap's key, there can only be one occurrence of the specific id in all our database, which can have multiple Taxpayers. That can cause multiple problems such as:

- If the receipt id already exists in a Taxpayer, it can't be added to any other Taxpayer.
- If the receipt id exists in multiple Taxpayers, it can only be successfully deleted from one of them.

The problem was solved by removing the HashMap receiptOwnerTRN and using the receiptHashMap of each Taxpayer when dealing with Receipt objects.

Furthermore, the infoFileHashMap was added in the TaxpayerManager class so the user can load info files regardless of the file's path. The updateInfoFile method updates only the specific file that the Taxpayer was loaded from and not all the info files that have the same tax registration number, which was the functionality of the updateFiles method of the legacy app.

PACKAGE INCOMETAXCALCULATOR.DATA.IO

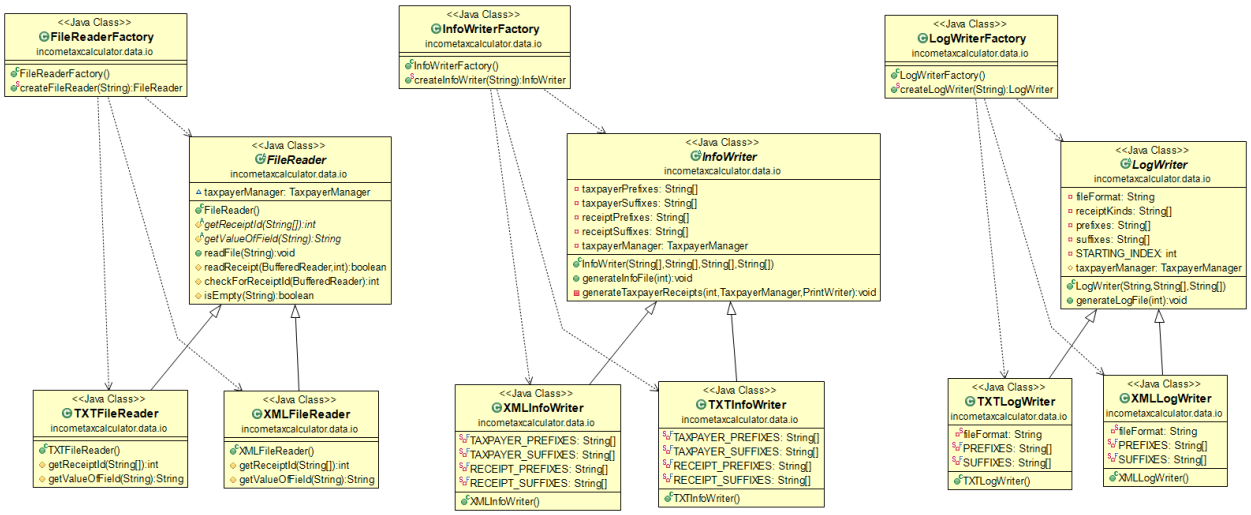

Figure 4: UML Class Diagram of the Package incometaxcalculator.data.io

The **TXTFileReader** and **XMLFileReader** class contained Duplicate Code in the checkForReceipt method. Therefore, the parts of the code that are different where extracted in the subclasses in the method getReceiptId. The checkForReceipt method was pulled up to the FileReader Class and renaimed to checkForReceiptId in order to be more concise on the functionality of the method.

The **FileReader** class had a Middle Man problem. The methods createTaxpayer and createReceipt simply delegate calls to the respective methods of the TaxpayerManager. Consequently, they were removed and the calls were done directly to the TaxpayerManager methods.
In addition, the comments that described the exceptions that the readFile method throws were removed because they are excess.

The **FileWriter** class had a severe Middle Man problem. All of the methods, except the generateFile() which is used by its subclasses, delegate calls to respective methods of the TaxpayerManager. Another problem of the class is Refuse Bequest, since some methods are used only by some subclasses. A final problem is that the class is very general. It groups together writers that update existing info files and writers that create log files.

Firstly, the FileWriter class was removed and replaced by two new classes, the InfoWriter and the LogWriter, and the 4 subclasses of the FileWriter were divided into the two new classes according to type of file (info/log) that they deal with.

The **InfoWriter** class has two subclasses, the **TXTInfoWriter** and the **XMLInfoWriter**. Both subclasses use the method generateFile and the only difference is in the constant string tags. Therefore, the following steps were taken:

- use array fields in the InfoWriter class for storing the sting tags
- initialize the arrays in the subclasses
- change the method's body to use data structures
- transfer the implementation of the method in the InfoWriter Class and remove it from the subclasses

The above was implemented in the LogWriter method and its subclasses, TXTLogWriter and XMLLogWriter.

Furthermore, the generateFile in the InfoWriter was renamed to generateInfoFile and the generateFile in the LogWriter was renamed to generateLogFile in order to be more specific about the functionality of the method.

PACKAGE INCOMETAXCALCULATOR.EXCEPTIONS

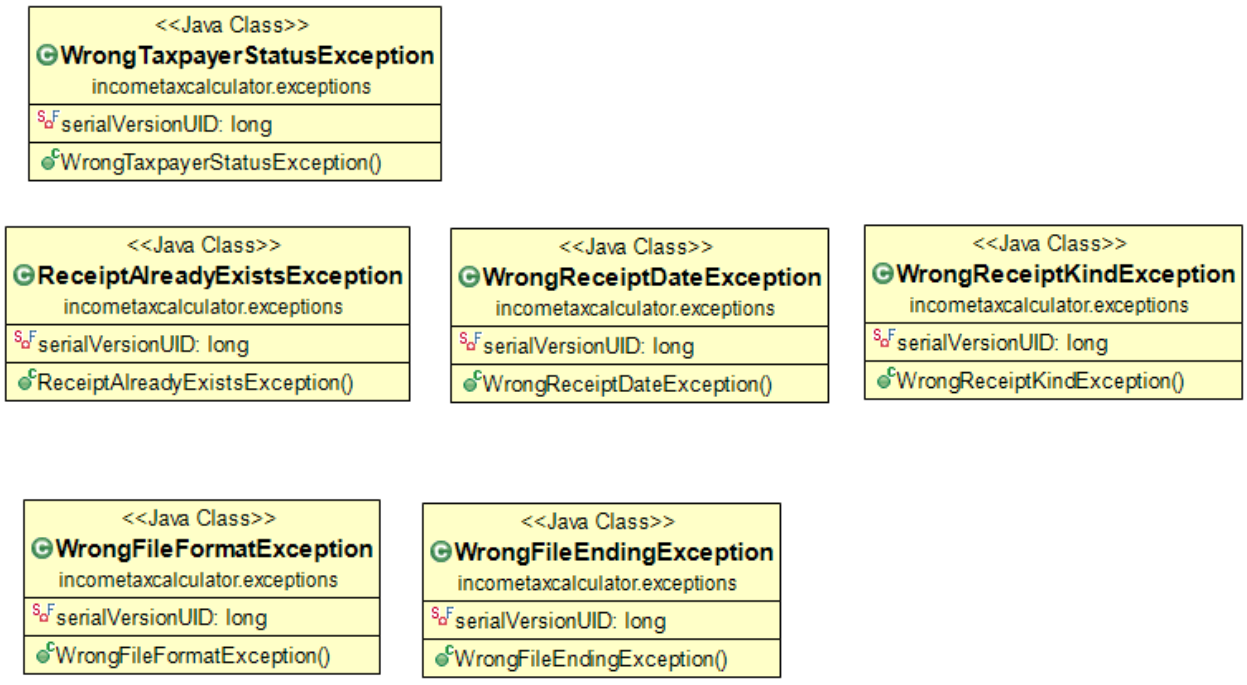

Figure 5: UML Class Diagram of the Package incometaxcalculator.exceptions

Regarding the package incometaxcalculator.exceptions there were no changes.

PACKAGE INCOMETAXCALCULATOR.GUI

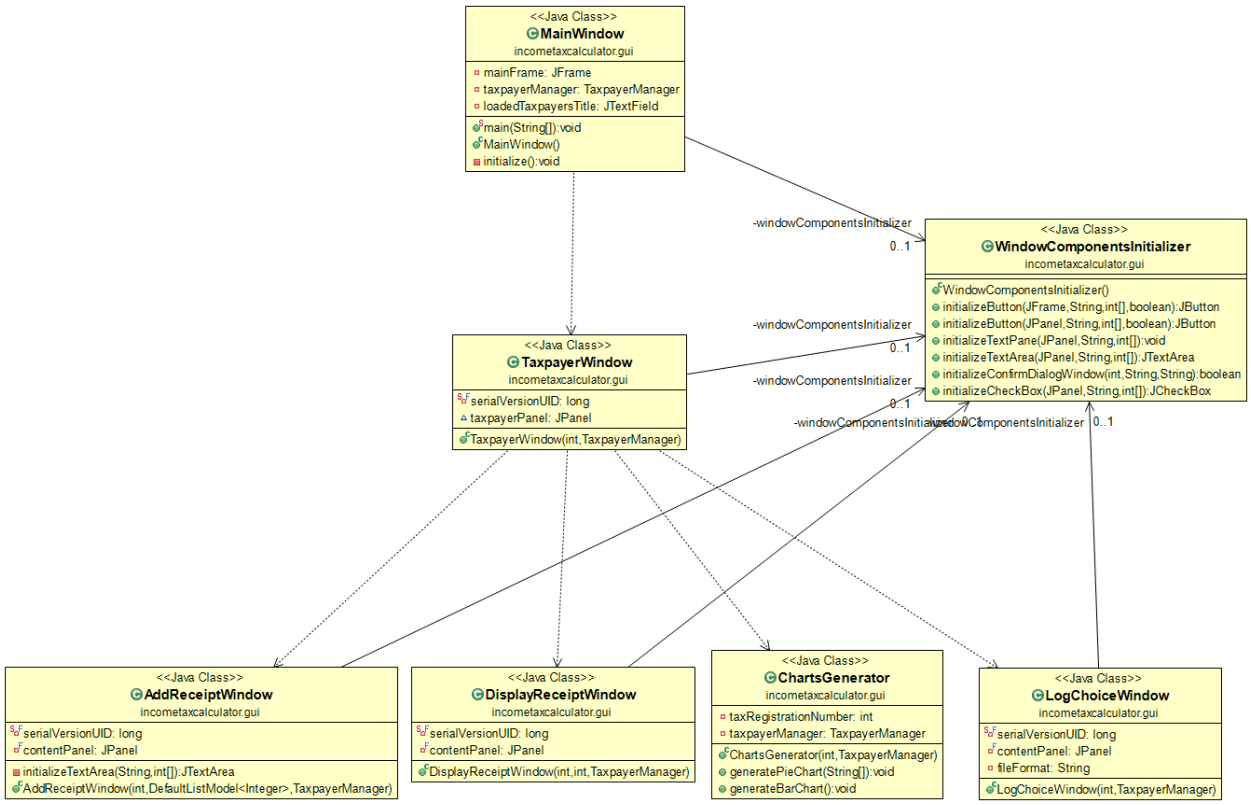

Figure 6: UML Class Diagram of the Package incometaxcalculator.gui

The GUI of the legacy application had several issues that made it hard to use. Therefore, it was completely replaced by a brand new one. The new GUI was written in Swing.

The **MainWindow** class is responsible for the main menu window, and it has the same functionality as the legacy menu. The user can Load, Select or Delete a Taxpayer.
Loading a Taxpayer is easier because the user chooses the info file from a file browser, so they don't have to type the file's name in.

In addition, the selection and deletion of a Taxpayer don't require the user to type in the registration number as it happened in the legacy app. The user just clicks on the Taxpayer's registration number from the main window's list and then clicks the 'Select Taxpayer' or 'Delete Taxpayer' button.
The 'Select Taxpayer' opens the Taxpayer window that is created in the TaxpayerWindow.

The **TaxpayerWindow** class creates a window that opens when the user selects a Taxpayer. The TaxpayerWindow has the same functionalities as in the legacy app and a newly added one that displays the content of the receipt that the user selected. The window displays the information of the selected Taxpayer and has the following buttons:

- 'Add Receipt'. The window that pops up is created in the **AddReceiptWindow** class.
  The fields of the receipt's details contain grey prompts that help the user understand the format of what they need to fill out.
- 'Display Receipt'. It is a new functionality of the GUI that uses the **DisplayReceiptWindow** class to create a new window. It displays the details of a Taxpayer's receipt.
- 'Delete Receipt'. In contrast to the legacy GUI where the user had to type in the receipt's id, the user just has to click on the receipt's id from the Taxpayer's list of receipts.
- 'Generate Charts'. This button has replaced 'View Report' button of the legacy app. The bar and pie chart that are generated use the xchart library instead of the jfreechart and they are more eye pleasing and correct in the data representation. The code for the charts is in the **ChartsGenerator** class.
- 'Export Log File'. This button has replaced 'Save Data' button of the legacy app. The user is prompted in a new window to choose the file format of the log file. The new window is created by the **LogChoiceWindow** class.

Furthermore, the new GUI has much cleaner code, since components, such as buttons, text areas, checkboxes, confirmation dialog windows, are initialized by methods in the **WindowComponentsInitializer** class.

Some problems that the new GUI failed to address is that:

- If the user has opened a Taxpayer window and then proceeds to delete the Taxpayer from the main menu window, the Taxpayer window still remains open but no button operation is possible.
- The user can open multiple windows for the same Taxpayer. The changes that happen to one window, such as the deletion or addition of a receipt, aren't refreshed to the rest windows.
- The Pie chart displays only the percentage and not the total amount of each kind of receipt.

CLASSES RESPONSIBILITIES AND COLLABORATIONS (CRC CARDS)

Package Name:	incometaxcalculator.data.management
Class Name:	Address
Responsibilities	**Collaborations**
The class is responsible for storing a company's address.	The class is associated with the Company class.

Package Name:	incometaxcalculator.data.management
Class Name:	Company
Responsibilities	**Collaborations**
The class is responsible for storing the details of a receipt's company.	The class is associated with the following classes: • Address • Receipt

Package Name:	incometaxcalculator.data.management
Class Name:	Receipt
Responsibilities	**Collaborations**
The class is responsible for storing the details of a receipt.	The class is associated with the following classes: • Company • Taxpayer • TaxpayerManager

Package Name:	incometaxcalculator.data.management
Class Name:	Taxpayer
Responsibilities	**Collaborations**
The class is responsible for storing the details of a Taxpayer. It calculates the Taxpayer's tax and it adds or deletes receipts from the Taxpayer.	The class is associated with the following classes: • Receipt • TaxpayerFactory • SingleTaxpayer • MarriedFillingJointlyTaxpayer • MarriedFillingSeparatelyTaxpayer • HeadOfHouseholdTaxpayer • TaxpayerManager

Package Name:	incometaxcalculator.data.management
Class Name:	TaxpayerFactory
Responsibilities	**Collaborations**
The class is responsible for creating different types of Taxpayers.	The class is associated with the following classes: • Taxpayer • SingleTaxpayer • MarriedFillingJointlyTaxpayer • MarriedFillingSeparatelyTaxpayer • HeadOfHouseholdTaxpayer • TaxpayerManager

Package Name:	incometaxcalculator.data.management
Class Name:	SingleTaxpayer
Responsibilities	**Collaborations**
The class is responsible for creating a Taxpayer whose marital status is Single.	The class is associated with the following classes: • Taxpayer • TaxpayerFactory

Package Name:	incometaxcalculator.data.management
Class Name:	MarriedFillingJointlyTaxpayer
Responsibilities	**Collaborations**
The class is responsible for creating a Taxpayer whose marital status is Married and fills jointly the tax declaration.	The class is associated with the following classes: • Taxpayer • TaxpayerFactory

Package Name:	incometaxcalculator.data.management
Class Name:	MarriedFillingSeparatelyTaxpayer
Responsibilities	**Collaborations**
The class is responsible for creating a Taxpayer whose marital status is Married and fills separately the tax declaration.	The class is associated with the following classes: • Taxpayer • TaxpayerFactory

Package Name:	incometaxcalculator.data.management
Class Name:	HeadOfHouseholdTaxpayer
Responsibilities	**Collaborations**
The class is responsible for creating a Taxpayer whose marital status is Head of Household.	The class is associated with the following classes: • Taxpayer • TaxpayerFactory

Package Name:	incometaxcalculator.data.management
Class Name:	TaxpayerManager
Responsibilities	**Collaborations**
The class is responsible for managing the loaded Taxpayers. It performs the: • loading and deletion of Taxpayers, • addition and deletion of receipts from a certain taxpayer, • updating of the info file and • exporting of a log file	The class is associated with the following classes: • Receipt • TaxpayerFactory • Taxpayer • Main Window • TaxpayerWindow • AddReceiptWindow • DisplayReceiptWindow • ChartsGenerator • LogChoiceWindow • FileReaderFactory • FileReader • InfoWriterFactory • InfoWriter • LogWriterFactory • LogWriter

Package Name:	incometaxcalculator.data.io
Class Name:	FileReader
Responsibilities	**Collaborations**
The class is responsible for reading the contents of an info file and loading them to the Taxpayer Manager.	The class is associated with the following classes: • FileReaderFactory • TaxpayerManager

Package Name:	incometaxcalculator.data.io
Class Name:	FileReaderFactory
Responsibilities	**Collaborations**
The class is responsible for creating different types of File Readers.	The class is associated with the following classes: • FileReader • TaxpayerManager • TXTFileReader • LogFileReader

Package Name:	incometaxcalculator.data.io
Class Name:	TXTFileReader
Responsibilities	**Collaborations**
The class is responsible for creating a TXT File Reader.	The class is associated with the following classes: • FileReaderFactory • FileReader

Package Name:	incometaxcalculator.data.io
Class Name:	XMLFileReader
Responsibilities	**Collaborations**
The class is responsible for creating an XML File Reader.	The class is associated with the following classes: • FileReaderFactory • FileReader

Package Name:	incometaxcalculator.data.io
Class Name:	InfoWriter
Responsibilities	**Collaborations**
The class is responsible for updating the info file of a loaded Taxpayer.	The class is associated with the following classes: • InfoWriterFactory • TaxpayerManager • TXTInfoWriter • XMLInfoWriter

Package Name:	incometaxcalculator.data.io
Class Name:	InfoWriterFactory
Responsibilities	**Collaborations**
The class is responsible for creating different types of Info Writers.	The class is associated with the following classes: • InfoWriter • TaxpayerManager • TXTInfoWriter • XMLInfoWriter

Package Name:	incometaxcalculator.data.io
Class Name:	TXTInfoWriter
Responsibilities	**Collaborations**
The class is responsible for creating a TXT Info Writer.	The class is associated with the following classes: • InfoWriterFactory • InfoWriter

Package Name:	incometaxcalculator.data.io
Class Name:	XMLInfoWriter
Responsibilities	**Collaborations**
The class is responsible for creating an XML Info Writer.	The class is associated with the following classes: • InfoWriterFactory • InfoWriter

Package Name:	incometaxcalculator.data.io
Class Name:	LogWriter
Responsibilities	**Collaborations**
The class is responsible for creating a log file for a loaded Taxpayer.	The class is associated with the following classes: • LogWriterFactory • TaxpayerManager • TXTLogWriter • XMLLogWriter

Package Name:	incometaxcalculator.data.io
Class Name:	LogWriterFactory
Responsibilities	**Collaborations**
The class is responsible for creating different types of Log Writers.	The class is associated with the following classes: • LogWriter • TaxpayerManager • TXTLogWriter • XMLLogWriter

Package Name:	incometaxcalculator.data.io
Class Name:	TXTLogWriter
Responsibilities	**Collaborations**
The class is responsible for creating a TXT Log Writer.	The class is associated with the following classes: • LogWriterFactory • LogWriter

Package Name:	incometaxcalculator.data.io
Class Name:	XMLLogWriter
Responsibilities	**Collaborations**
The class is responsible for creating an XML Log Writer.	The class is associated with the following classes: • LogWriterFactory • LogWriter

Package Name:	incometaxcalculator.gui
Class Name:	MainWindow
Responsibilities	**Collaborations**
The class is responsible for creating the main menu window.	The class is associated with the following classes: • WindowComponentsInitializer • TaxpayerWindow • TaxpayerManager

Package Name:	incometaxcalculator.gui
Class Name:	TaxpayerWindow
Responsibilities	**Collaborations**
The class is responsible for creating a window for a selected Taxpayer. This class: • displays the Taxpayers information, • adds new receipts, • displays receipts' information, • deletes receipts, • generates charts and • produces a log file.	The class is associated with the following classes: • WindowComponentsInitializer • MainWindow • AddReceiptWindow • DisplayReceiptWindow • ChartsGenerator • LogChoiceWindow • TaxpayerManager

Package Name:	incometaxcalculator.gui
Class Name:	AddReceiptWindow
Responsibilities	**Collaborations**
The class is responsible for adding a new receipt.	The class is associated with the following classes: • WindowComponentsInitializer • TaxpayerWindow • TaxpayerManager

Package Name:	incometaxcalculator.gui
Class Name:	DisplayReceiptWindow
Responsibilities	**Collaborations**
The class is responsible for displaying the details of a receipt.	The class is associated with the following classes: • WindowComponentsInitializer • TaxpayerWindow • TaxpayerManager

Package Name:	incometaxcalculator.gui
Class Name:	ChartsGenerator
Responsibilities	**Collaborations**
The class is responsible for generating a bar chart and a pie chart.	The class is associated with the following classes: • TaxpayerWindow • TaxpayerManager

Package Name:	incometaxcalculator.gui
Class Name:	LogChoiceWindow
Responsibilities	**Collaborations**
The class is responsible for creating a log file.	The class is associated with the following classes: • WindowComponentsInitializer • TaxpayerWindow • TaxpayerManager

Package Name:	incometaxcalculator.gui
Class Name:	WindowComponentsInitializer
Responsibilities	**Collaborations**
The class is responsible for initializing swing components.	The class is associated with the following classes: • MainWindow • TaxpayerWindow • AddReceiptWindow • DisplayReceiptWindow • LogChoiceWindow