INCOME TAX CALCULATOR

OVERALL REPORT

VERSION 1.0

Andreas Triantafyllopoulos 4504

Foteini Yfanti 4516

TABLE OF CONTENTS

INTRODUCTION

The point of this project is to modify an income tax calculator. This calculator distinguishes the types of tax payers based on their family status and their expenses based on its type. The software uses a user friendly GUI which can read information from txt and xml files, make changes to each tax payer, update their information and create modified LOG files.

REFACTORED DESIGN

USE CASES

Name	Load Data
Use case ID	1
Actors	User
Pre conditions	1. To exist the file <TaxRegistationNumber>_INFO.txt or *.xml.
Main flow of events	1. The use case starts when the user clicks «Load Taxpayer» button. 2. Then the user navigates to the location containing the desired file. 3. The user selects the file and clicks the «Open» option.
Alternative flow 1	1. The use case starts when the user clicks «Load Taxpayer» button. 2. Then the user navigates to the location containing the desired file. 3. The user selects the file and clicks the «Open» option. 4. The selected file has not the appropriate format (file title or file type or file contents).
Alternative flow 2	1. The use case starts when the user clicks «Load Taxpayer» button. 2. The user clicks on the «Cancel» button and the window closes.
Post conditions	1. The file data are loaded. 2. The tax registration number is included in the tax registration number list.

Name	Select Tax Payer
Use case ID	2
Actors	User
Pre conditions	1. The tax payer's data have to be loaded. 2. The tax payer's tax registration number has to be included in the tax registration number list.
Main flow of events	1. The use case starts when the user double clicks on the preferable tax registration number.
Post conditions	1. A new window shows up that contains the tax payer's data, the receipts and other functions to add, delete, save or present additional data.

Name	Add New Receipt
Use case ID	3
Actors	User
Pre conditions	1. The tax payer has to be selected. 2. The window containing the tax payer's data has to be on display.
Main flow of events	1. The use case starts when the user clicks on the «Add Receipt» button. 2. A new window shows up with gaps on the receipt fields, so the user can add a new receipt to the tax payer's data.
Alternative flow 1	1. The use case starts when the user clicks «Add Receipt» button. 2. The user does not know the correct format of the date. 3. The user clicks on the «Information» button.

	4. A new window appears with the corrects format of date. 5. The user clicks on the «Click to Close Info Window!» to close it. 6. Fills the gaps properly and clicks on «OK» button to save the new receipt.
Alternative flow 2	1. The use case starts when the user clicks «Add Receipt» button. 2. The user types wrong kind of information in a cell (such as String when a number is required). 3. The program outputs an error message to inform the user.
Post conditions	1. The new receipt is added to the list of receipts.

Name	Delete Receipt
Use case ID	4
Actors	User
Pre conditions	1. The window containing the tax payer's data has to be on display. 2. The receipt has to exist.
Main flow of events	1. The use case starts when the user clicks on the «Delete Receipt» button. 2. A new window shows up in order to type which receipt ID the user wants to delete. 3. The user clicks on «OK» button and the window closes.
Alternative flow 1	1. The use case starts when the user clicks on the «Delete Receipt» button. 2. A new window shows up in order to type which receipt ID the user wants to delete.

	3. The receipt ID that user types does not exist or mistyped. 4. The window closes and nothing is deleted.
Alternative flow 2	1. The use case starts when the user clicks on the «Delete Receipt» button. 2. A new window shows up in order to type which receipt ID the user wants to delete. 3. The user does not want to delete a receipt anymore. 4. The user clicks on the «Cancel» button and the window closes.
Post conditions	1. The receipt is deleted.

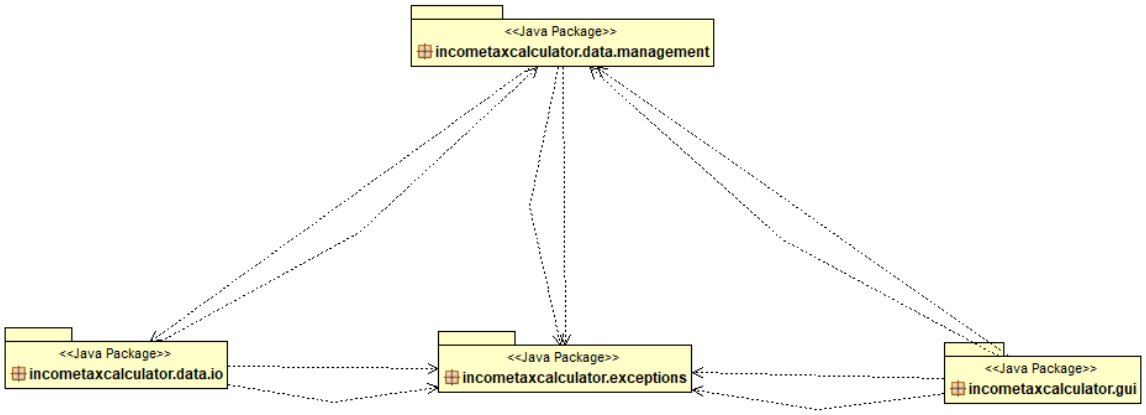
Name	View Report
Use case ID	5
Actors	User
Pre conditions	1. A list of receipts has to exist.
Main flow of events	1. The use case starts when the user clicks on the « View Report» button. 2. Two new windows show up in order to present the analytics to the user. The first one shows the percentage of the total amount of each kind of receipts and the second one shows the tax analysis in $ in bar charts. 3. The user clicks on «X» buttons on each window and the windows close.
Post conditions	1. The report charts are being presented.

Name	Save Data
Use case ID	6
Actors	User
Pre conditions	1. The window containing the tax payer's data has to be on display.
Main flow of events	1. The use case starts when the user clicks on the «Save Data» button. 2. A new window appears in order to navigate to the location that the user wants to save the LOG file. 3. The user selects the type of file that wants to be exported to. 4. The user clicks on «Save» button and the window closes.
Alternative flow 1	1. The use case starts when the user clicks on the «Save Data» button. 2. A new window appears in order to navigate to the location that the user wants to save the LOG file. 3. The user does not want to save the changes anymore and clicks on «Cancel» button, so the window closes.
Post conditions	1. The LOG file is saved.

Name	Delete Taxpayer
Use case ID	7
Actors	User
Pre conditions	1. The tax payer has to be loaded.
Main flow of events	1. The use case starts when the user clicks on the «Delete Taxpayer» button. 2. A new window appears in order to type the tax registration number of the tax payer that the user wants to delete.

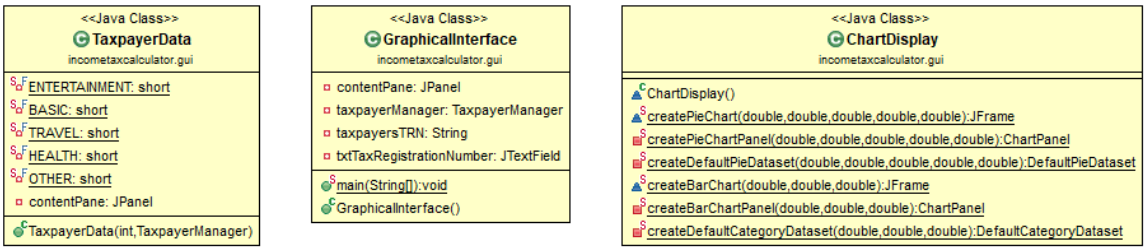
	3. The user clicks on «OK» button and the window closes.
Alternative flow 1	1. The use case starts when the user clicks on the «Delete Taxpayer» button. 2. A new window appears in order to type the tax registration number of the tax payer that the user wants to delete. 3. The user does not want to delete the taxpayer anymore and clicks on «Cancel» button, so the window closes.
Post conditions	1. The tax payer is deleted.

ARCHITECTURE

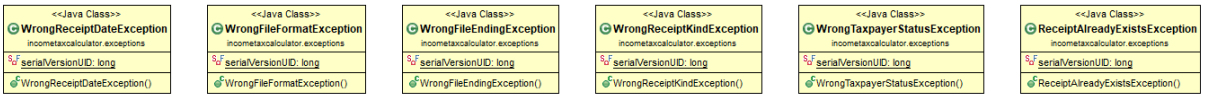


DETAILED DESIGN

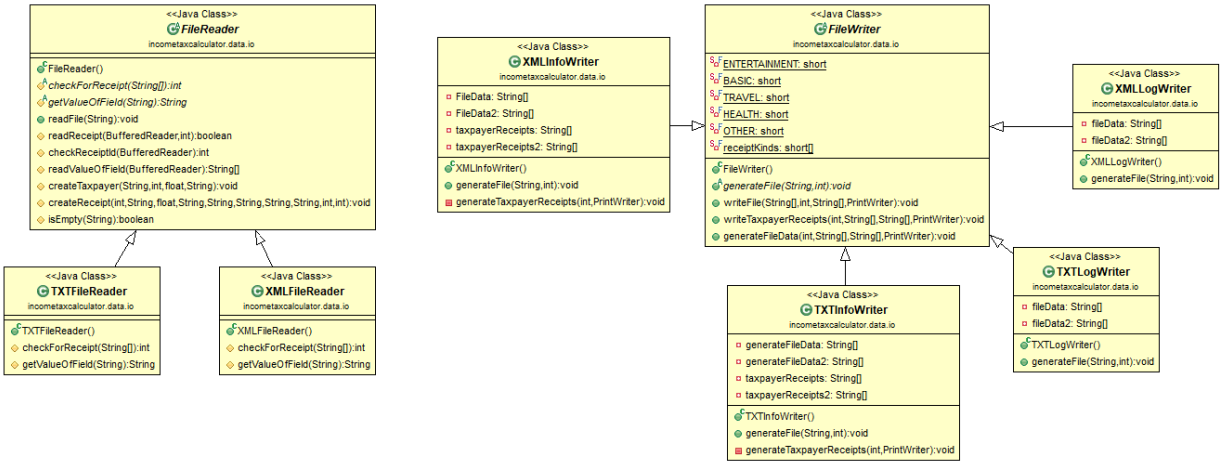
IncomeTaxCalculator.GUI



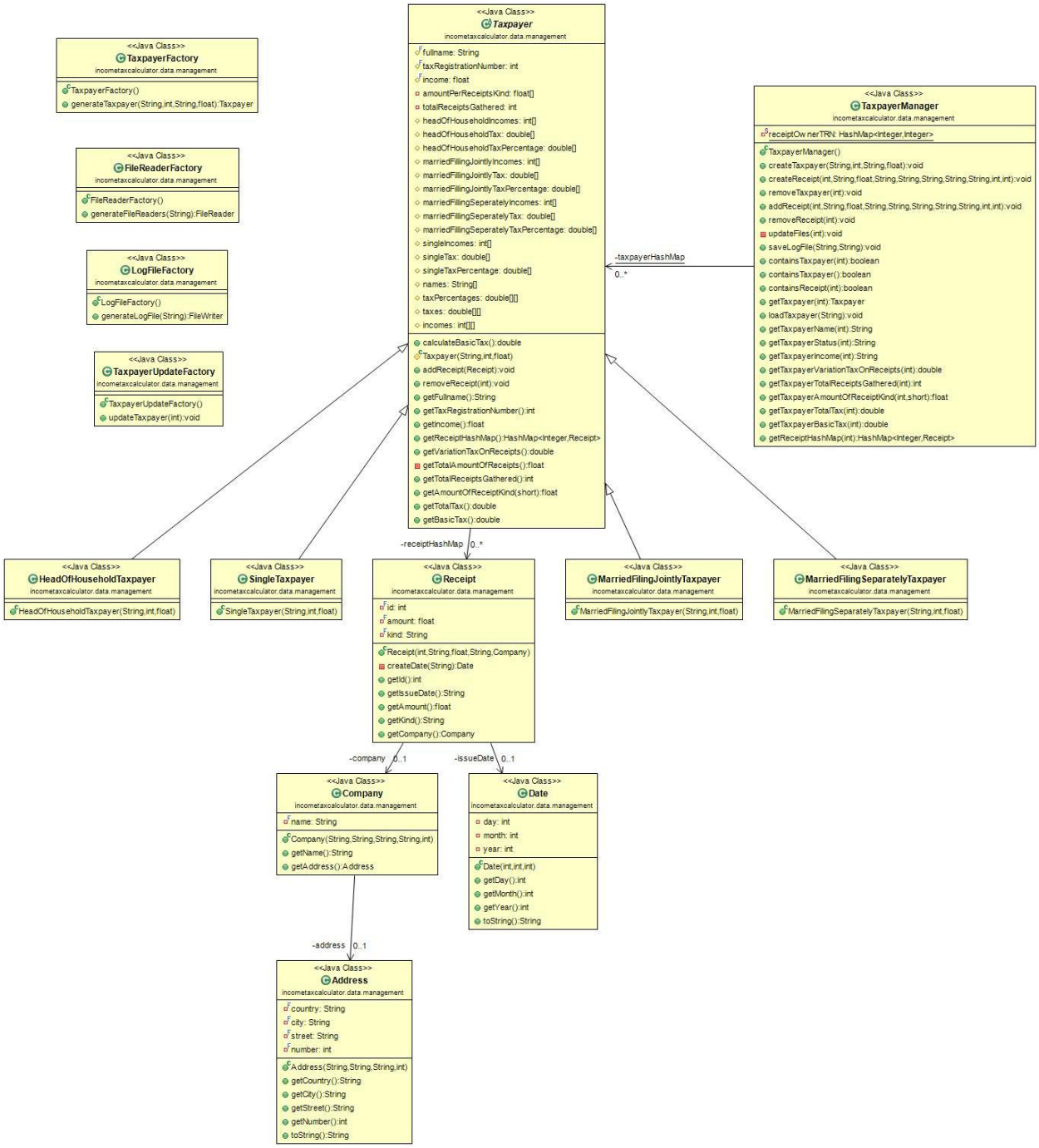
IncomeTaxCalculator.Exceptions



IncomeTaxCalculator.Data.IO



IncomeTaxCalculator.data.management



Company class:

We removed the following methods because they were never used to access data: getCountry(), getCity(), getStreet() and getNumber(). We also changed the return type of getAddress() to Address, so we can access the data of an address object.

Taxpayer class:

We replaced the if-else logic in the addReceipt method with a for loop that repeatedly checks a list that contains the receipt's kinds, and if the criteria is met we add that amount to the corresponding index of the amountPerReceiptsKind array.

We replaced the if-else logic in the removeReceipt method with a for loop in the same way we did with addReceipt above.

We replaced the if-else logic in the getVariationTaxOnReceipts with a for loop in the same way we did with addReceipt.

We added a new method called calculateBasicTax in order to reduce the duplicate code in the classes that extend the Taxpayer class. We needed to create a few helping arrays that contain the classified data for every type of tax payer.

SingleTaxpayer, MarriedFilingJointly, MarriedFilingSeperately, HeadOfHousehold class:

We removed the calculateBasicTax method because it contained code similar to all of these classes and we created a generic method called calculateBasicTax which we added to Taxpayer class.

For each class (type of tax payer) we added the tax variables that are used by the state of Minnesota in specific arrays.

TaxpayerManager class:

We removed the conditional logic from the createTaxpayer method and placed in a new parameterized factory called TaxpayerFactory.

We removed the conditional logic from the updateFiles method and placed in a new parameterized factory called TaxpayerUpdateFactory.

We removed the conditional logic from the saveLogFile method and placed in a new parameterized factory called LogFileFactory.

We removed the conditional logic from the loadTaxpayer method and placed in a new parameterized factory called FileReaderFactory.

TXTFileReader and XMLFileReader classes:

We removed the duplicate code from checkForReceipt and getValueOfField methods and placed it in checkReceiptId and readValueOfField methods of FileReader class.

FileReader class:

We removed the unnecessary lines that read every line from input stream and created a new method called readValueOfField that autonomously reads from input stream.

FileWriter class:

We removed all the getter methods as they were unnecessary, removed the delegating methods and used directly the TaxpayerManager methods inside the class's methods.

We added writeTaxpayerReceipts and generateFileData methods so that we can write both types of files with one method. So every method gets the core of the type of file from the object it refers to and adds to it the data from TaxpayerManager.

TXTInfoWriter and XMLInfoWriter classes:

We removed the file writing commands from their methods and delegated it to their superclass FileWriter.

TXTLogWriter and XMLLogWriter classes:

Similar to TXTInfoWriter and XMLInfoWriter classes, we removed the file writing commands from their methods and delegated it to their superclass FileWriter.

GraphicalInterface class:

We made the software more functional, so now the user can load an input file from any directory using a file navigating environment. The file chooser can accept only txt and xml files, which are the types of files our software can handle, and there is no need to specify the type of file in the beginning or the need to type the TRN.

We added the ability to double click a loaded TRN and open the Taxpayer's Data window, instead of typing the TRN ourselves in the window that appears after clicking «Select Taxpayer».

TaxpayerData class:

We extended the functionality of the software by helping the user add a new receipt to a tax payer. The software now helps by letting the user choose one of the given kinds for the new receipt from a drop-down menu. The software also has an information button that presents the user with the correct way of typing-in the Date of the receipt. To close the window we added a «Click to Close Info Window!» button. We also added the ability for the user to select the directory where the LOG file will be saved. We also added the ability to choose the type of file that will be exported in the directory navigator as a drop down menu with the options «XML files» and «TXT files».

CLASSES RESPONSIBILITIES AND COLLABORATIONS (CRC CARDS)

Class Name: FileReader(Abstract)	
Responsibilities	**Collaborations**
▪ This class is responsible for reading the input xml or txt file the user choose. o The class reads the given info for the tax payer and all the receipts that belongs to him. o The class creates a Manager object and adds a new Taxpayer object to it with the info it reads from input file. o The class creates Receipt objects based on the input file and adds them to the Manager	▪ The class associates with TXTFileReader and XMLFileReader classes as they extend it. ▪ The class associates with TaxpayerManager class as it creates TaxpayerManager objects.

object.	

Class Name: FileWriter(Abstract)	
Responsibilities	**Collaborations**
▪ This class is responsible for writing the files we need to save. o The class writes the info that is contained in the TaxpayerManager object for a certain tax payer, to his new LOG file. o The class writes the receipts added from the user, to the INFO file of the tax payer.	▪ The class associates with TXTLogWriter, TXTInfoWriter, XMLLogWriter and XMLInfoWriter classes as they extend it. ▪ The class associates with TaxpayerManager class as it reads info from TaxpayerManager objects. It also associates with Receipt and Address classes as it uses them to get information for the tax payer.

Class Name: TXTFIleReader	
Responsibilities	**Collaborations**
▪ This class is responsible for reading a txt input file.	▪ The class associates with FileReader class as it extends it.

Class Name: XMLFIleReader	
Responsibilities	**Collaborations**
▪ This class is responsible for reading a XML input file.	▪ The class associates with FileReader class as it extends it.

Class Name: TXTInfoWriter	
Responsibilities	**Collaborations**
▪ This class is responsible for writing the changes to the receipts of the tax payer in TXT format.	▪ The class associates with FileWriter class as it extends it.

Class Name: TXTLogWriter	
Responsibilities	**Collaborations**
▪ This class is responsible for writing the new Log file of the tax payer in TXT format.	▪ The class associates with FileWriter class as it extends it.

Class Name: XMLInfoWriter	
Responsibilities	**Collaborations**

▪ This class is responsible for writing the changes to the receipts of the tax payer in XML format.	▪ The class associates with FileWriter class as it extends it.

Class Name: XMLLogWriter	
Responsibilities	**Collaborations**
▪ This class is responsible for writing the new Log file of the tax payer in XML format.	▪ The class associates with FileWriter class as it extends it.

Class Name: Address	
Responsibilities	**Collaborations**
▪ This class is responsible for holding the information for a specific address of a Company.	

Class Name: Company	
Responsibilities	**Collaborations**
▪ This class is responsible for holding the	▪ The class associates with Address class as it stores the address information in

information of a company.	an Address object.

Class Name: Date	
Responsibilities	**Collaborations**
▪ This class is responsible for holding the information of a receipt's date.	

Class Name: FileReaderFactory	
Responsibilities	**Collaborations**
▪ This class is responsible for creating the appropriate kind of reader object (xml or txt) for the input files.	▪ The class associates with XMLFileReader and TXTFileReader.

Class Name: HeadOfHouseholdTaxpayer	
Responsibilities	**Collaborations**
▪ This class is responsible for holding the information for the calculation of the head of household type of tax payer.	

Class Name: MarriedFilingJointlyTaxpayer	
Responsibilities	**Collaborations**
▪ This class is responsible for holding the information for the calculation of the married filing jointly type of tax payer.	

Class Name: MarriedFilingSeperatelyTaxpayer	
Responsibilities	**Collaborations**
▪ This class is responsible for holding the information for the calculation of the married filing seperately type of tax payer.	

Class Name: SingleTaxpayer	
Responsibilities	**Collaborations**
▪ This class is responsible for holding the information for the calculation of the single type of tax payer.	

Class Name: LogFileFactory	
Responsibilities	**Collaborations**
▪ This class is responsible for creating the appropriate kind of FileWriter object (xml or txt) for the output files.	▪ The class associates with XMLLogWriter and TXTLogWriter.

Class Name: Receipt	
Responsibilities	**Collaborations**
▪ This class is responsible for holding the information of a receipt.	▪ This class associates with class Date and class Company as it creates and holds objects of these types.

Class Name: Taxpayer	
Responsibilities	**Collaborations**
▪ This class is responsible for calculating the taxes based on the type of the tax payer, his income and his expenses. ▪ This class is responsible for holding the information of a tax payer. o It holds the receipts a tax	▪ This class associates with class Receipt as it gets info from receipt objects and it lists them.

payer has. o It holds the incomes and the taxes of the tax payer and the minnesota tax percentages. ▪ This class is responsible for removing receipts from a tax payer. ▪ This class is responsible for adding receipts to a tax payer.	

Class Name: TaxpayerFactory	
Responsibilities	**Collaborations**
▪ This class is responsible for creating the appropriate kind of Taxpayer object (Married Filing Jointly, Married Filing Separately, Head of Household or Single) for the info of the loaded tax payer.	▪ The class associates with MarriedFilingJointlyTaxpayer, MarriedFilingSeparatelyTaxpayer, HeadOfHouseholdTaxpayer and SingleTaxpayer classes.

Class Name: TaxpayerManager	
Responsibilities	**Collaborations**
▪ This class is responsible for handling	▪ The class associates with

the user's commands based on the use cases of the software. ▪ This class is responsible for holding the list of the tax payers that are loaded and their TRN.	TaxpayerFactory, Receipt, Company, Taxpayer, LogFileFactory, TaxpayerUpdateFactory and FileReaderFactory classes as it creates or generally uses their objects.

Class Name: TaxpayerUpdateFactory	
Responsibilities	**Collaborations**
▪ This class is responsible for creating the appropriate type of FileWriter object (XMLInfoWriter and TXTInfoWriter) to write the new info of the loaded tax payer.	▪ The class associates with XMLInfoWriter and TXTInfoWriter classes.

Class Name: ChartDisplay	
Responsibilities	**Collaborations**
▪ This class is responsible for creating and displaying two types of charts (Pie chart and Bar chart)	

Class Name: GraphicalInterface	
Responsibilities	**Collaborations**
▪ This class is responsible for displaying the main window's buttons and functionality while handling any errors that occur.	▪ The class associates with TaxpayerManager and TaxpayerData classes.

Class Name: TaxpayerData	
Responsibilities	**Collaborations**
▪ This class is responsible for displaying the tax payer's window's buttons and functionality while handling any errors that occur. ▪ This class is responsible for handling the addition of a new receipt from the user. ▪ This class is responsible for handling the delete of a receipt from the user. ▪ This class is responsible for presenting the tax payer data to the user as a chart.	▪ The class associates with TaxpayerManager and Receipt classes.

▪ This class is responsible for saving the calculated data to a new Log file.	

Class Name: TaxpayerUpdateFactory	
Responsibilities	**Collaborations**
▪ This class is responsible for creating the appropriate type of FileWriter object (XMLInfoWriter and TXTInfoWriter) to write the new info of the loaded tax payer.	▪ The class associates with XMLInfoWriter and TXTInfoWriter classes.