MINESSOTA INCOME TAX CALCULATOR

REENGINEERING THE LEGACY CODE

OVERALL REPORT

VERSION 1.0

Evangelos Tzortzis, AM: 3088

TABLE OF CONTENTS

INTRODUCTION

The objectives of this phase of the project are to refactor the code in order to fix some issues, as well as to reengineer some parts of the GUI with an updated and easier to use interface and the addition of new interface options that were missing from the previous implementation.

REFACTORED DESIGN

USE CASES

Use case ID	UC1: LoadTaxpayerInfoFile
Actors	User
Pre conditions	-
Main flow of events	1. The use case starts when the user presses the button from the UI to load a taxpayer file. 2. The user chooses the file he wishes to open by browsing the directories 3. The system loads the file into the program.
Alternative flow 1	2. The user can exit/cancel the menu anytime.
Alternative flow 2	3. If the registration number format of the file name is not 9 digits: a. The system notifies the user. b. The system prompts the user to select a file again.
Alternative flow 3	3. If the taxpayer exists already within the program:

	a. The system notifies the user. b. The operation is cancelled and the use case is terminated.
Alternative flow 4	3. If the file does not exist OR the file format is wrong OR the file ending is wrong OR the taxpayer status is wrong OR the kind of a receipt is wrong: a. The system notifies the user. b. The operation is cancelled and the use case is terminated.
Post conditions	The file is loaded into the program.

Use case ID	UC2: SelectTaxpayer
Actors	User
Pre conditions	A taxpayer must be loaded into the system.
Main flow of events	1. The use case starts when the user presses the button to select a taxpayer. 2. The user selects the file he wishes from a drop-down list menu of loaded taxpayers. 3. The system opens a new window menu with the details of the selected taxpayer.
Alternative flow 1	1. The user can exit/cancel the selection menu anytime.
Post conditions	A new window appears with detailed information about the file.

Use case ID	UC3: DeleteTaxpayer
Actors	User
Pre conditions	A taxpayer must be loaded into the system.
Main flow of events	1. The use case starts when the user presses the button to delete a taxpayer. 2. The user selects the file he wishes from a drop-down list menu of loaded taxpayers. 3. The system deletes the taxpayer from the program.
Alternative flow 1	1. The user can exit/cancel the delete menu anytime.
Post conditions	The taxpayer is removed from the program.

Use case ID	UC4: AddReceipt
Actors	User
Pre conditions	A taxpayer must be loaded into the system and selected by the user.
Main flow of events	1. The use case starts when the user presses the button to add a receipt. 2. The user inputs all the fields. 3. The system adds the receipt to the taxpayer and updates the info file.
Alternative flow 1	1. The user can exit/cancel the menu anytime.
Alternative flow 2	2. If a field is invalid or the receipt ID already exists: a. The system notifies the user.

	b. The use case terminates.
Post conditions	The receipt is added to the taxpayer and the taxpayer file is updated.

Use case ID	UC5: DeleteReceipt
Actors	User
Pre conditions	A taxpayer must be loaded into the system and selected by the user.
Main flow of events	1. The use case starts when the user presses the button to delete a receipt. 2. The user selects the receipt he wishes to delete from a drop-down list of existent receipt IDs. 3. The system deletes the receipt of the taxpayer and updates the info file.
Alternative flow 1	2. The user can exit/cancel the selection anytime.
Post conditions	The receipt is removed from the taxpayer and the taxpayer file is updated.

Use case ID	UC6: ViewReport
Actors	User
Pre conditions	A taxpayer must be loaded into the system and selected by the user.
Main flow of	1. The use case starts when the user presses the button to view report.

events	2. The system displays a bar graph with tax info and a pie chart with the amount of money per receipt kind.
Post conditions	A bar graph and a pie chart appear.

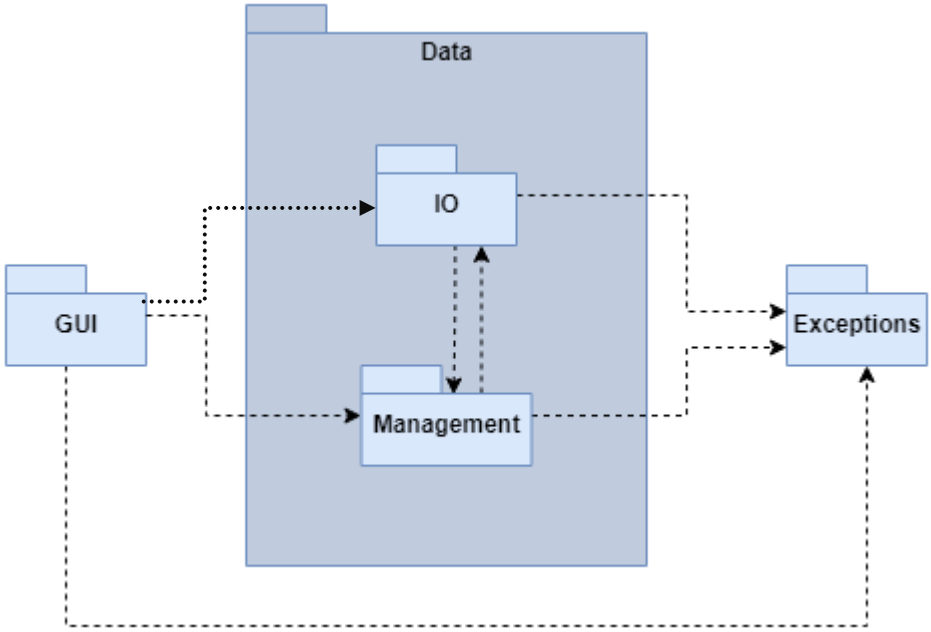
Use case ID	UC7: SaveData
Actors	User
Pre conditions	A taxpayer must be loaded into the system and selected by the user.
Main flow of events	1. The use case starts when the user presses the button to save a log file for the taxpayer. 2. The user selects the file type (xml or txt) and the directory in which to save the log file. 3. The system saves the log file with a summary for the taxpayer.
Alternative flow 1	1. The user can exit/cancel the menu anytime.
Post conditions	A log and/or a txt file are saved in the specified directory

Use case ID	UC8: ChangeTaxpayerStatus
Actors	User
Pre conditions	A taxpayer must be loaded into the system and selected by the user.
Main flow of events	1. The use case starts when the user presses the button to edit the taxpayer status. 2. The user inputs the new status from a drop-down menu with the available status categories.

	3. The system changes the taxpayer status and updates the info file.
Alternative flow 1	1. The user can exit/cancel the menu anytime.
Post conditions	The status is changed, and the taxpayer info file is updated.

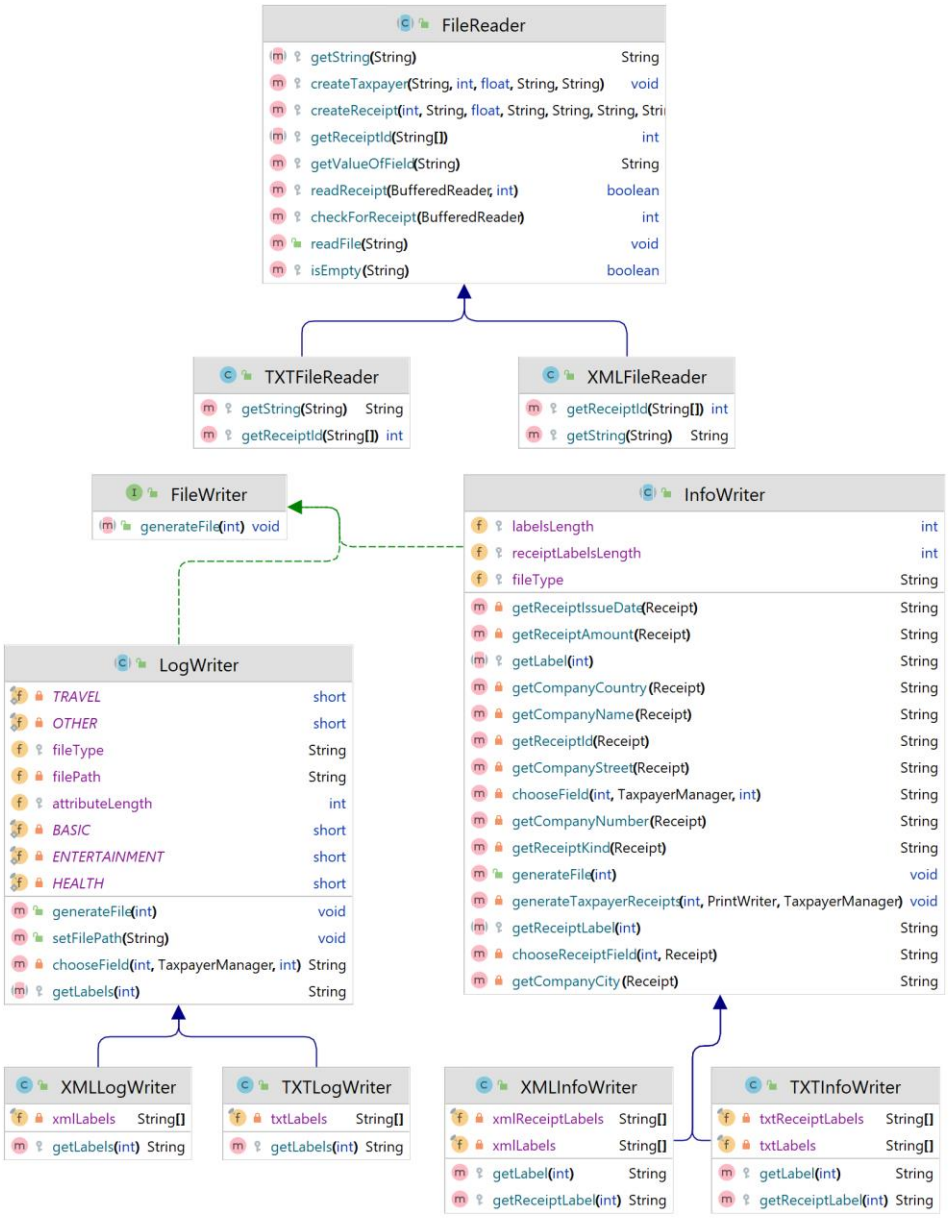
Use case ID	UC9: ChangeTaxpayerIncome
Actors	User
Pre conditions	A taxpayer must be loaded into the system and selected by the user.
Main flow of events	1. The use case starts when the user presses the button to edit the taxpayer income. 2. The user inputs the new income value. 3. The system changes the taxpayer status and updates the info file.
Alternative flow 1	1. The user can exit/cancel the menu anytime.
Alternative flow 2	2. If a income is invalid: c. The system notifies the user. d. The use case terminates.
Post conditions	The income is changed, and the taxpayer info file is updated.

ARCHITECTURE

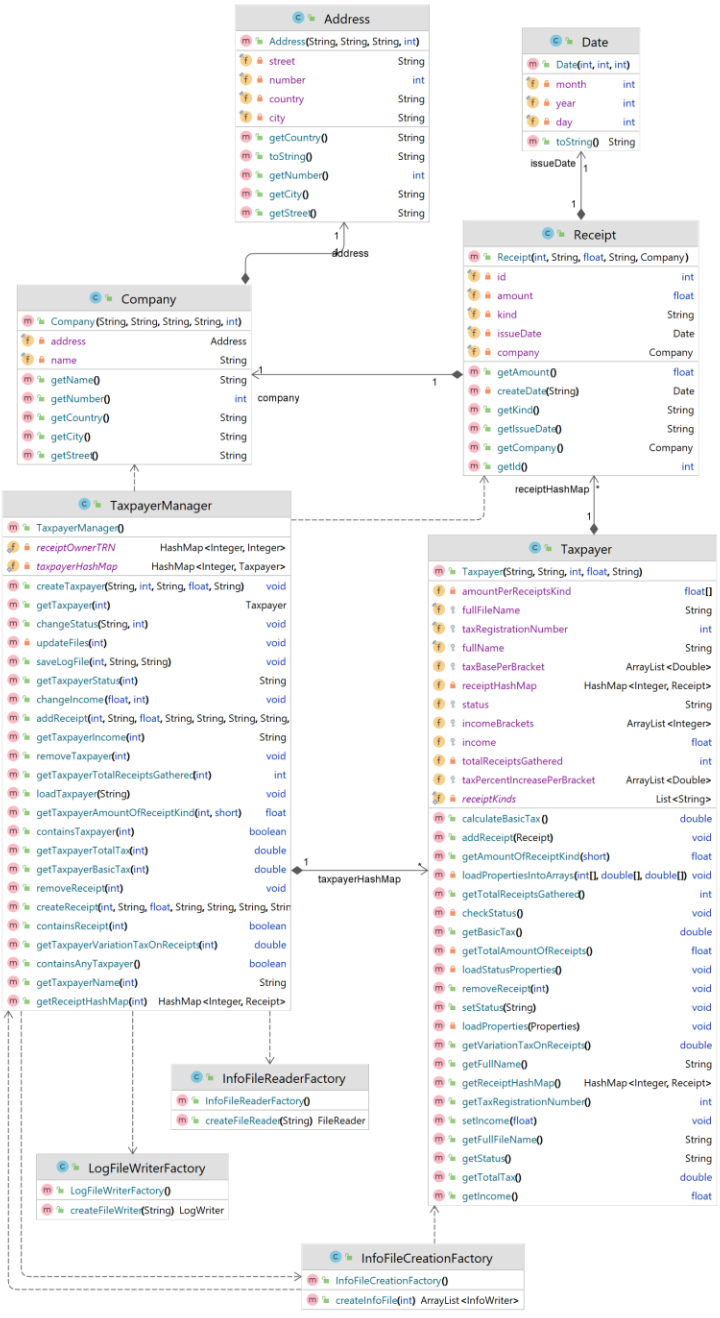


DETAILED DESIGN

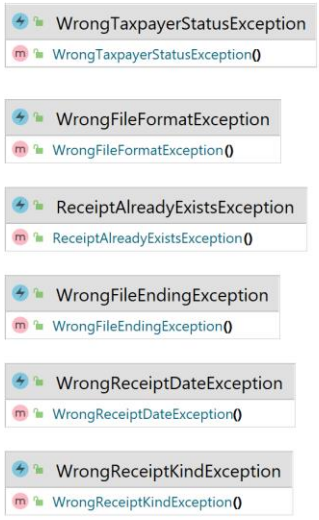
data.io:



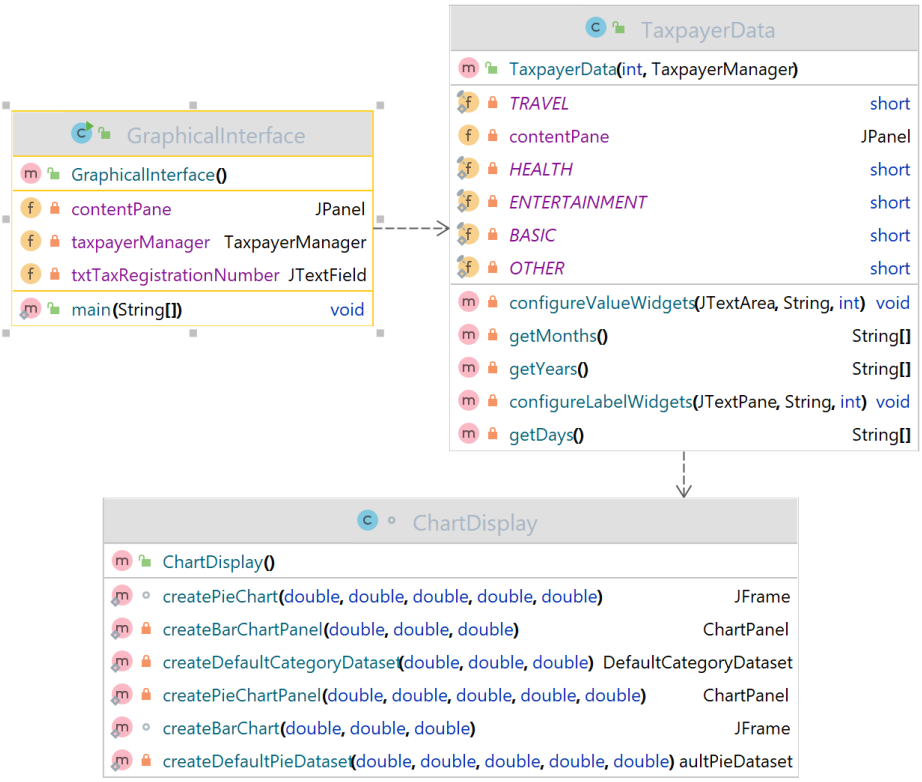
data.management:



exceptions:



gui:



The old design contained structures within the project's classes that were problematic and had to be addressed. This phase consists of the refactoring and limited reengineering of the project. Specifically:

1. Some classes had unused methods that could be removed.

    - From the class Company the method getAddress() was removed.

    - From the class Date the methods getDay(), getMonth(), getYear() were removed.

2. The Taxpayer class methods addReceipt(), removeReceipt(), getVariationTaxOnReceipts() had extended conditional logic that has been simplified with the use of for loop in the case of getVariationTaxOnReceipts() or with the use of the method contains in addReceipt(), removeReceipt() which further reduces the lines of code.

3. The subclasses of the Taxpayer class that implemented different categories of taxpayers had methods that were performing similar operations, with the only difference being some constants like income brackets, tax base per bracket and tax percent increase per bracket. To mitigate this initially the common method calculateBasicTax() was moved to the Taxpayer class and each subclass initialized the values of some protected lists upon an object creation. After adding to the GUI the option to edit the category of a taxpayer, the subclasses had to be removed. In their place a configuration file of name "taxpayerStatusConfig" in the folder Config was created. This file loads the specific to the taxpayer class attribute lists upon Taxpayer object creation, as well as changes the values if the taxpayer category is changed. Additionally, the implementation of calculateBasicTax() was shortened by using a for loop instead of the chained if statements.

4. The TaxpayerManager class is very large and contains multiple responsibilities. To simplify the class, some responsibilities are delegated to subordinate classes.

a) Initially the logic of the createTaxpayer() method was moved to a factory that created the type of the taxpayer based on the category that was read from the INFO file. As discussed above though, to allow for the category to be changed within the app, each taxpayer is created as a Taxpayer object with the status being a parameter in the constructor that can change through the GUI.

b) The method updateFiles() updates the INFO files in case a receipt is added/removed and with the updated specification there are changes to the status/income of the taxpayer. In this method there was conditional logic that created the writers to the INFO files and then generated the files for a taxpayer. The creation of different types of writer objects in txt or xml format were moved to the factory class InfoFileCreationFactory() within the package incometaxcalculator.data.management.factories.

c) Similarly the saveLogFile() has conditional logic for the creation of log files with xml or txt format. The factory class LogFileWriterFactory() creates the log file to which the data are written based on the selected file format.

d) The method loadTaxpayer() loads an INFO file taking as input the format of the file. Likewise the previous cases, the type of reader was changed to be created in a factory named InfoFileReaderFactory() instead of inside the method.

5. The classes TXTFileReader, XMLFileReader had duplicate code in some parts. The method checkForReceipt() was extracted to the base FileReader class and the parts that were different were extracted as new shorter methods and defined as abstract in it with implementations in each subclass. In a comparable way, the method getString() is defined as abstract in the base class.

6. The class FileWriter acted as a middleman for the TaxpayerManager class. For this issue the FileWriter class was converted to an Interface and its delegating methods were removed and the TaxpayerManager methods now are called directly in the subclasses that implement it, like InfoWriter and LogWriter. Moreover, some methods are used only by one subclass of the two, so they were pushed down to classes that use them only.

7. The algorithms that produce the output files from the classes TXTInfoWriter and XMLInfoWriter are similar. A new abstract class InfoWriter is made where all the common methods are implemented as well as some abstract methods and in each subclass only the lists of strings of same length that differ are initialized along with the implementation of the abstract methods. The generateFile() and generateTaxpayerReceipts() methods in InfoWriter were shortened substantially too with the use of a for loop for each attribute.

8. The classes TXTLogWriter and XMLLogWriter were modified the same way as the previous point with the creation of the abstract class LogWriter where the common methods are implemented and the initialization of the different lists in each subclass.

9. In addition to the above refactoring changes, some extra small changes were made like renaming some variables and methods with better names, extracting some parts of big methods to smaller private ones and changing the order/positioning of the methods within the classes in order to follow the guide about Class Organization from the book Clean Code, where it states that the methods should be arranged public first, then private utility ones that are used by the public method, then the next public method etc., following the stepdown rule so as to be read like a newspaper article. Some changes were made to the TaxpayerData gui class also to eliminate duplicate code in the creation of graphical elements. This was achieved with the methods configureLabelWidgets () and configureValueWidgets() that set some parameters which are the same for multiple elements.

10. Regarding the GUI, some changes were made to ease the use of the program and to introduce new capabilities to the user.

a. The “Load Taxpayer” action now, instead of requiring the user to input the type of INFO file and the taxpayer number, prompts the user for a file selection from the filesystem. This change also added the capability to select INFO files in different directories. The full file path is added as a parameter to the constructor of the Taxpayer class.

b. The actions to “Select Taxpayer” and “Delete Taxpayer” were changed from having to type the taxpayer registration number to drop-down menus of the loaded taxpayers with the appropriate message in case there is none.

c. In the taxpayer menu that appears after selecting one from the main menu multiple changes were made:

- An area was created in the window where the details of the receipts are displayed when a receipt number is clicked on.

- Edit buttons for the Status and Income properties for the taxpayer were added. When edited, the report and the log file change the INFO files are also updated automatically. For this reason changeStatus() and changeIncome() methods were added to the TaxpayerManaGer class that call setStatus() and setIncome() in the Taxpayer class and update the files.

- In the submenu window that appears when the “AddReceipt” is button is pressed, some options were changed. The Date now is a drop down menu, which eliminates the mistyping of the date in the wrong format. Additionally, the Kind attribute is a drop-down menu with the available receipt categories, so that input mistakes are stopped.

- The “DeleteReceipt” button was changed to a drop-down menu with the available receipt IDs for ease of use and the elimination of user input error.

- The option to save data was renamed to “Save Data To Log File” for better explanation of the operation. Furthermore, the menu had only selection of the log file format to be saved and not the directory, so a directory selection area was placed inside the menu window.

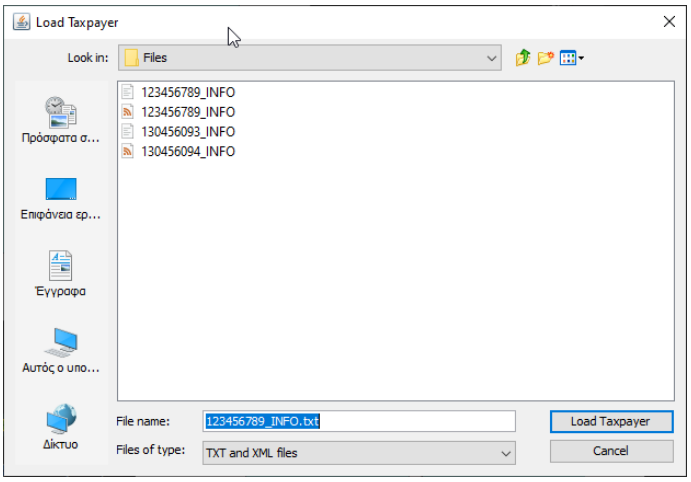
11. For testing purposes the class TaxpayerManagerTests was created in the package tests with multiple junit tests for the various operations of the program. Tests include:

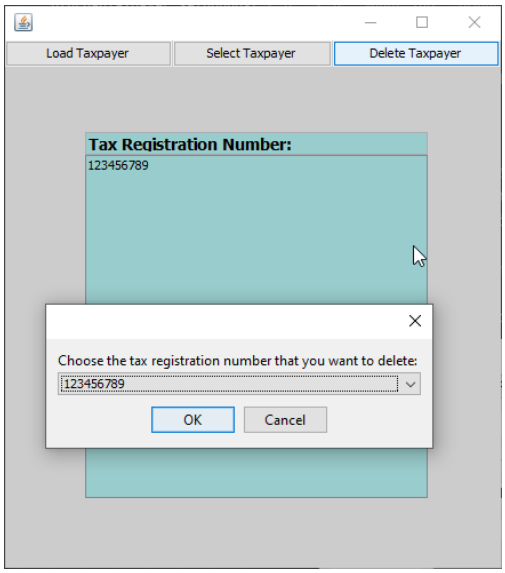
    - Loading taxpayer INFO file in txt format.
    - Loading taxpayer INFO file in xml format.
    - Removing a taxpayer successfully from the program.
    - Adding a receipt to the taxpayer, while also testing the exception in the case that the receipt already exists.
    - Removing a receipt from the taxpayer.
    - Saving a log file in both txt and xml formats with the information specified in the functional requirements.
    - Testing that the basic tax is calculated correctly.
    - Testing that the total tax is calculated correctly.
    - Testing that the taxpayer status change updates the status of the loaded taxpayer as well as the INFO file automatically.
    - Testing that the taxpayer income change updates the income of the loaded taxpayer as well as the INFO file automatically.

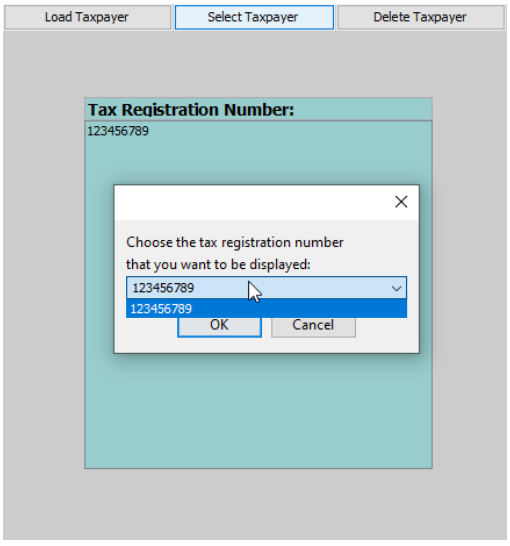
- Testing the various exceptions specified in the exceptions package regarding the loading of the taxpayer INFO file.

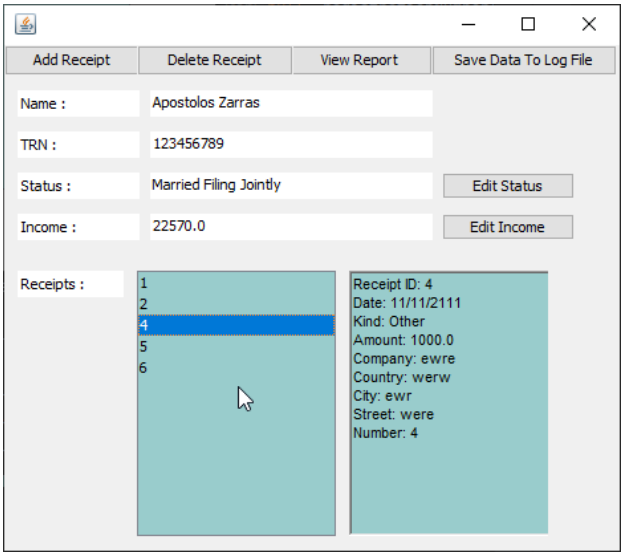
The taxpayer info files that were supplied are moved to the Files folder. The wrong files used to test the exceptions of loading exist in the testFiles folder.

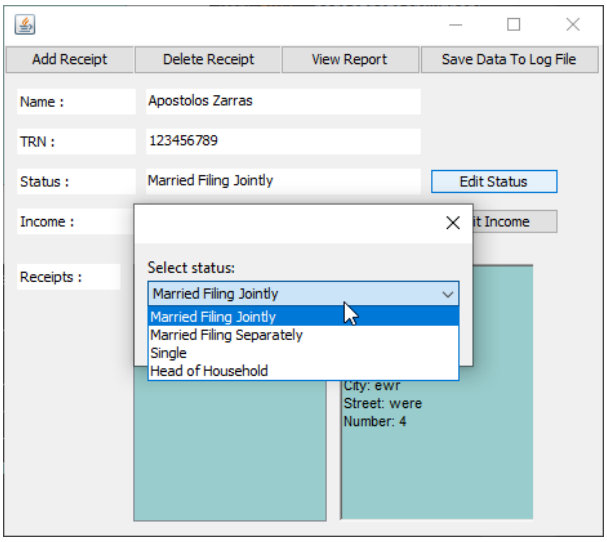
Below are shown some indicative screenshots from the application:

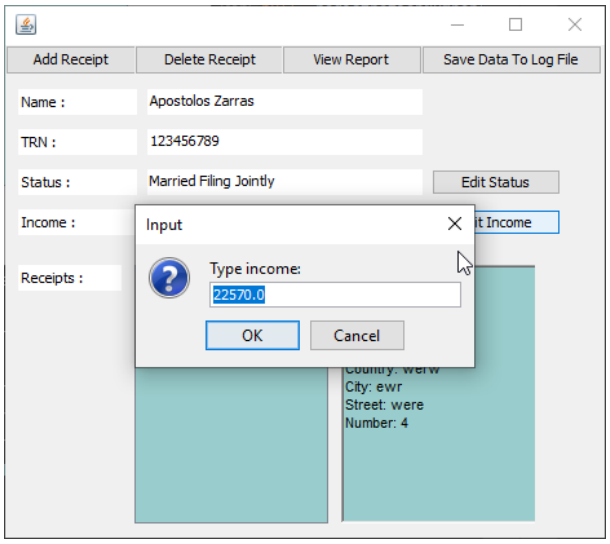


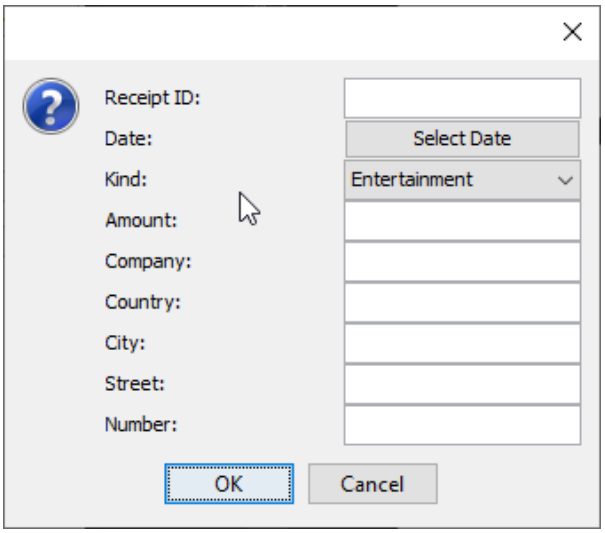


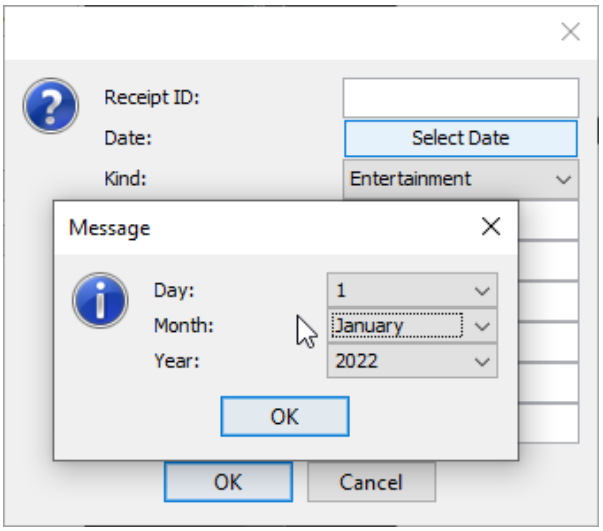


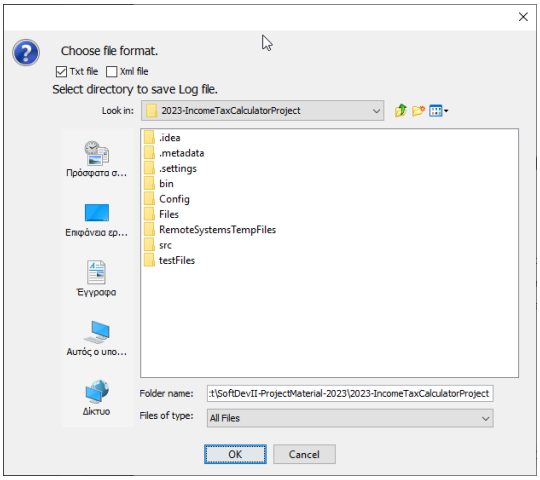


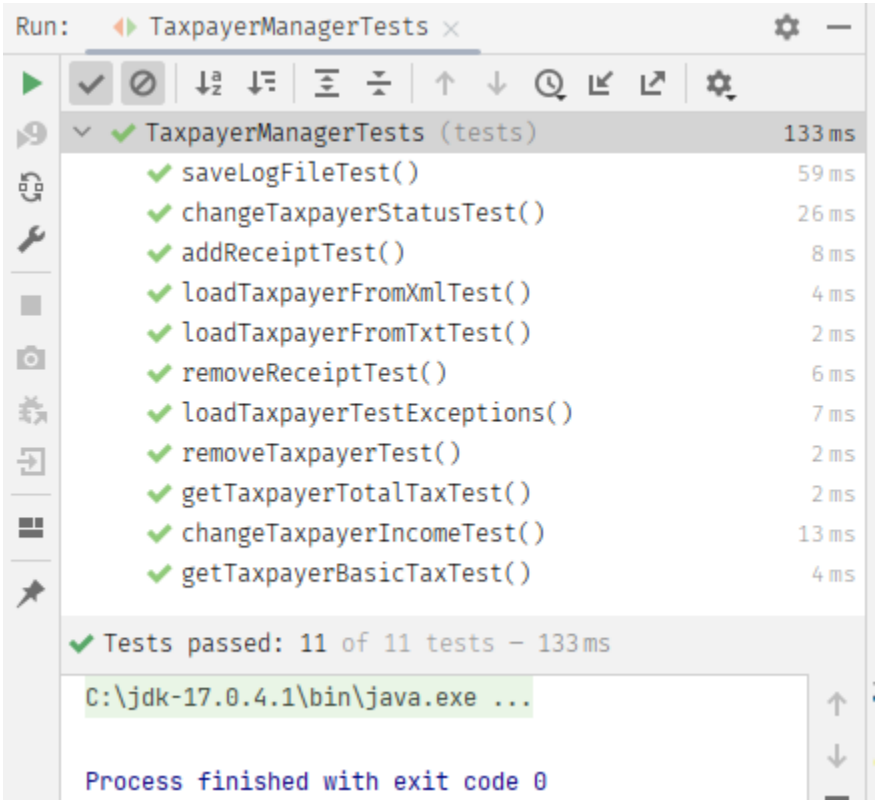

Add Receipt
Delete Receipt
View Report
Save Data To Log File
Name :
Apostolos Zarras
TRN :
123456789
Status :
Married Filing Jointly
Edit Status
Income :
Receipts :
Select status:
Married Filing Jointly
Married Filing Jointly
Married Filing Separately
Single
Head of Household
City: ewr
Street: were
Number: 4


Add Receipt
Delete Receipt
View Report
Save Data To Log File
Name :
Apostolos Zarras
TRN :
123456789
Status :
Married Filing Jointly
Edit Status
Income :
Receipts :
Input
Type income:
22570.0
OK
Cancel
City: ewr
Street: were
Number: 4


Receipt ID:
Date:
Select Date
Kind:
Entertainment
Amount:
Company:
Country:
City:
Street:
Number:
OK
Cancel


Receipt ID:
Date:
Select Date
Kind:
Entertainment
Message
Day:
1
Month:
January
Year:
2022
OK
OK
Cancel


Choose file format.
Txt file
Xml file
Select directory to save Log file.
Look in:
2023-IncomeTaxCalculatorProject
.idea
.metadata
.settings
bin
Config
Files
RemoteSystemsTempFiles
src
testFiles
Folder name:
Files of type:
All Files
OK
Cancel



Τρέξιμο του αρχείου TaxpayerManagerTests.java με τα junit test.

CLASSES RESPONSIBILITIES AND COLLABORATIONS (CRC CARDS)

Class Name:	
Responsibilities	**Collaborations**
▪ List the responsibilities of the class in simple precise statements - don't use passive voice.	▪ List the dependencies and associations of this class with the other classes of the project

○ E.g., This class is responsible for this task. ○ This class performs this activity...	

Class Name: FileReader	
Responsibilities	**Collaborations**
▪ This class performs the reading of the taxpayer INFO file. ▪ This class performs the loading the taxpayer INFO file.	▪ TXTFileReader ▪ XMLFileReader ▪ InfoFileReaderFactory ▪ TaxpayerManager
Class Name: TXTFileReader	
Responsibilities	**Collaborations**
▪ This class is responsible for getting the values from the taxpayer INFO file which is formatted in txt format. ▪ This class extends the FileReader abstract class.	▪ FileReader ▪ InfoFileReaderFactory

Class Name: XMLFileReader	
Responsibilities	**Collaborations**
▪ This class is responsible for getting the values from the taxpayer INFO file which is formatted in xml format. ▪ This class extends the FileReader abstract class.	▪ FileReader ▪ InfoFileReaderFactory

Class Name: InfoWriter	
Responsibilities	**Collaborations**
▪ This class overwrites information to the taxpayer INFO file.	▪ TXTInfoWriter ▪ XMLInfoWriter ▪ InfoFileCreationFactory ▪ TaxpayerManager

Class Name: TXTInfoWriter	
Responsibilities	**Collaborations**
▪ This class provides the txt specific	▪ InfoWriter

labels/tags to write to the taxpayer INFO file in txt format and ▪ extends the InfoWriter abstract class.	▪ InfoFileCreationFactory

Class Name: XMLInfoWriter	
Responsibilities	**Collaborations**
▪ This class provides the xml specific labels/tags to write to the taxpayer INFO file in txt format and ▪ extends the InfoWriter abstract class.	▪ InfoWriter ▪ InfoFileCreationFactory

Class Name: LogWriter	
Responsibilities	**Collaborations**
▪ This class writes taxpayer summary information to the taxpayer LOG file.	▪ TXTLogWriter ▪ XMLLogWriter ▪ LogFileWriterFactory ▪ TaxpayerManager

Class Name: XMLLogWriter	
Responsibilities	**Collaborations**
▪ This class provides the xml specific labels/tags to write to the taxpayer LOG file in xml format and ▪ extends the LogWriter abstract class.	▪ LogWriter ▪ LogFileWriterFactory

Class Name: TXTLogWriter	
Responsibilities	**Collaborations**
▪ This class provides the txt specific labels/tags to write to the taxpayer LOG file in txt format and ▪ extends the LogWriter abstract class.	▪ LogWriter ▪ LogFileWriterFactory

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

company where a receipt was produced.	

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

Class Name: Company	
Responsibilities	**Collaborations**
▪ This class is responsible for holding information about the company where a receipt was produced.	▪ Address ▪ Receipt ▪ TaxpayerManager

Class Name: Receipt	
Responsibilities	**Collaborations**
▪ This class is responsible for holding information about the company where	▪ Address

a receipt was produced.	▪ Receipt ▪ TaxpayerManager

Class Name: LogFileWriterFactory	
Responsibilities	**Collaborations**
▪ This factory class performs the creation of the log writers based on the selected format.	▪ LogWriter ▪ TXTLogWriter ▪ XMLLogWriter ▪ TaxpayerManager

Class Name: InfoFileCreationFactory	
Responsibilities	**Collaborations**
▪ This factory class performs the creation of the taxpayer info writers based on the selected format(s).	▪ InfoWriter ▪ TXTInfoWriter ▪ XMLInfoWriter ▪ TaxpayerManager

Class Name: InfoFileReaderFactory	
Responsibilities	**Collaborations**
▪ This factory class performs the creation of the info writers based on the selected format (txt of xml).	▪ FileReader ▪ TXTFileReader ▪ XMLFileReader ▪ TaxpayerManager

Class Name: Taxpayer	
Responsibilities	**Collaborations**
▪ This class is responsible for loading the status properties from the config file, ▪ adding receipts to a taxpayer, ▪ calculating the basic tax amount, ▪ calculating the total tax amount.	▪ TaxpayerManager ▪ Receipt

Class Name: TaxpayerManager	
Responsibilities	**Collaborations**

▪ This class performs loading the taxpayers to a hashmap, ▪ the receipt operations, ▪ the info file loading and updating, ▪ the saving to log files, ▪ the updating of the income and sstatus of taxpayers.	▪ LogWriter ▪ InfoWriter ▪ FileReader ▪ InfoFileReaderFactory ▪ InfoFileCreationFactory ▪ LogFileWriterFactory ▪ Taxpayer ▪ Receipt ▪ Company ▪ GraphicalInterface ▪ TaxpayerData

Class Name: GraphicalInterface	
Responsibilities	**Collaborations**
▪ This class is responsible for the main graphical interface window.	▪ TaxpayerManager ▪ TaxpayerData

Class Name: TaxpayerData	
Responsibilities	**Collaborations**
▪ This class is responsible for the taxpayer graphical interface window.	▪ GraphicalInterface ▪ TaxpayerManager ▪ Company ▪ Receipt

Class Name: ChartDisplay	
Responsibilities	**Collaborations**
▪ This class is responsible for the chart windows.	▪ GraphicalInterface