MINNESOTA INCOME TAX CALCULATION PROJECT

REENGINEERING THE LEGACY CODE

OVERALL REPORT

VERSION FINAL

Nikolaos Spyropoulos AM: 3077

Spyridon Tsotzolas AM: 3099

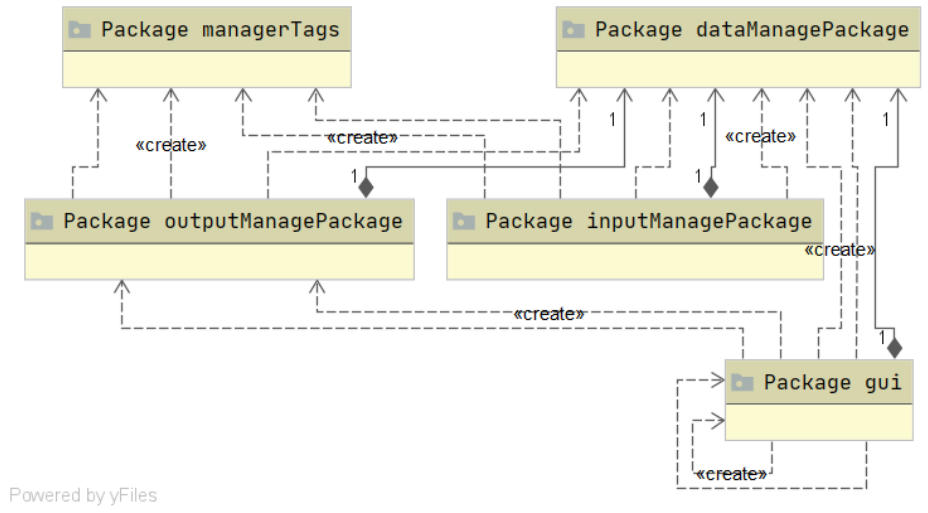
TABLE OF CONTENTS

INTRODUCTION

In this phase, after we study the given project, we spot some general problems which needed to be resolved. Having emplemented all the necessary refactoring of the initial code, we are ready to provide the architecture, detailed design and implementation of each class.

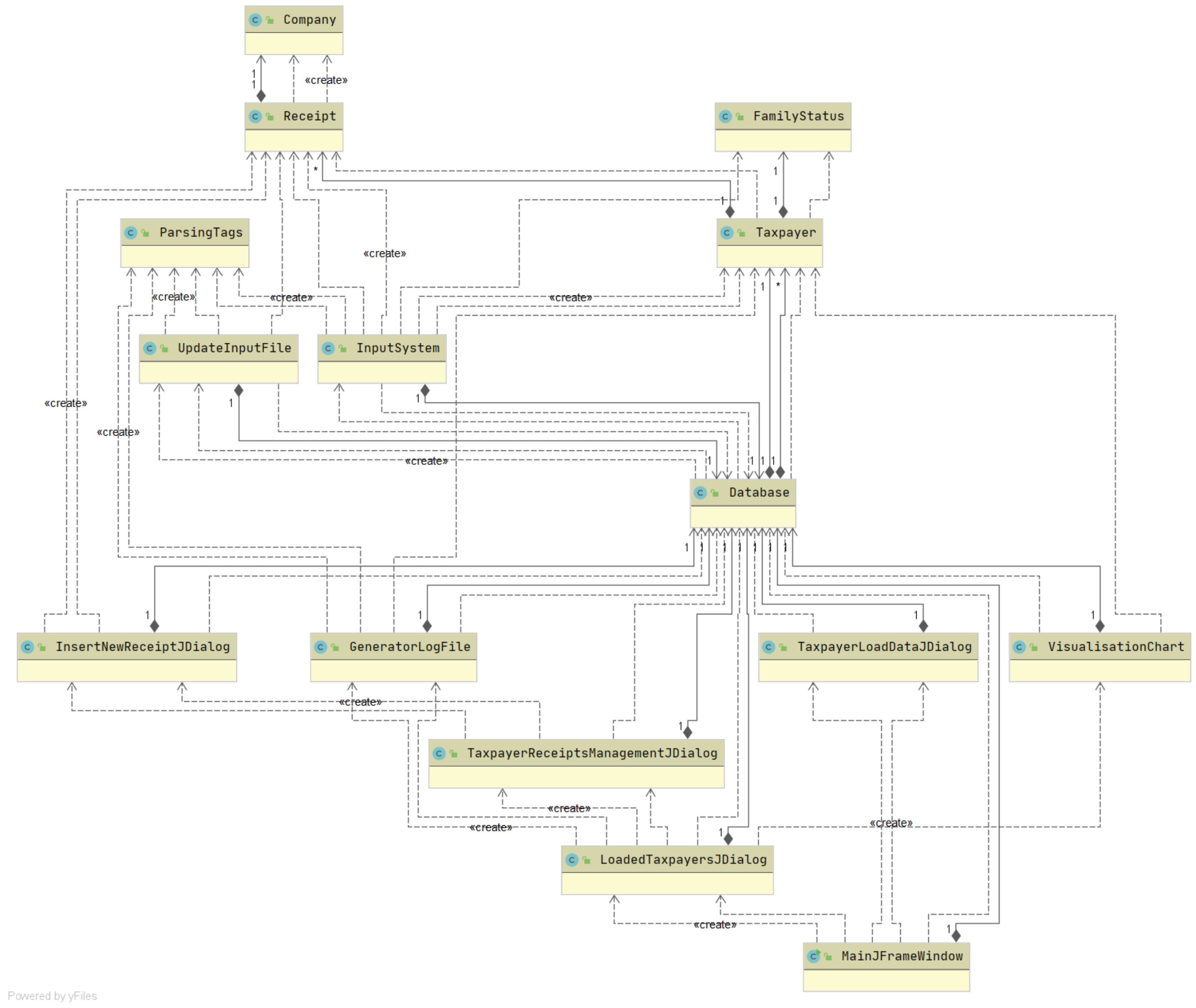
ARCHITECTURE

UML PACKAGE

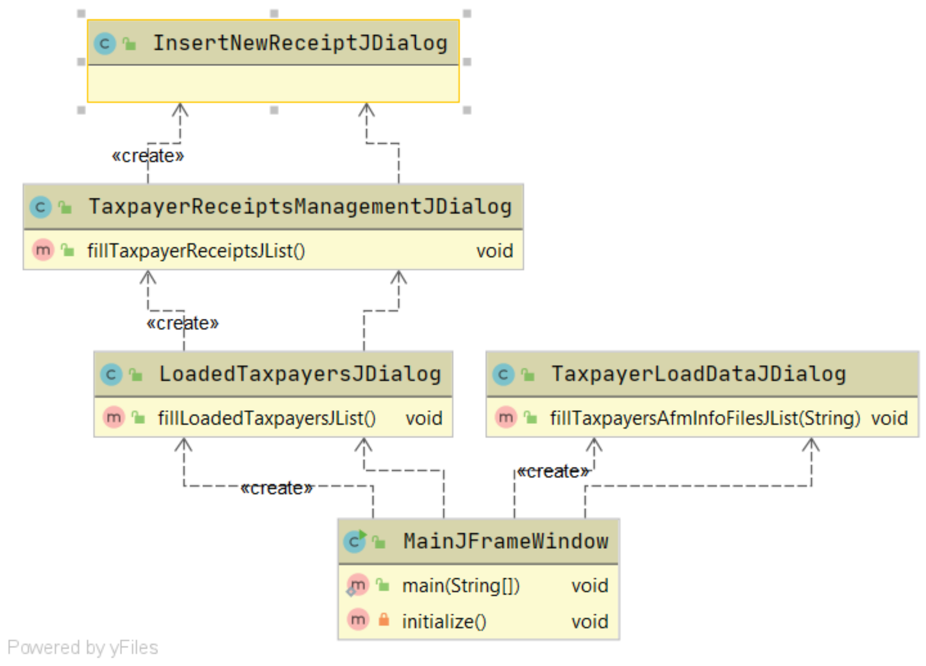


DETAILED DESIGN

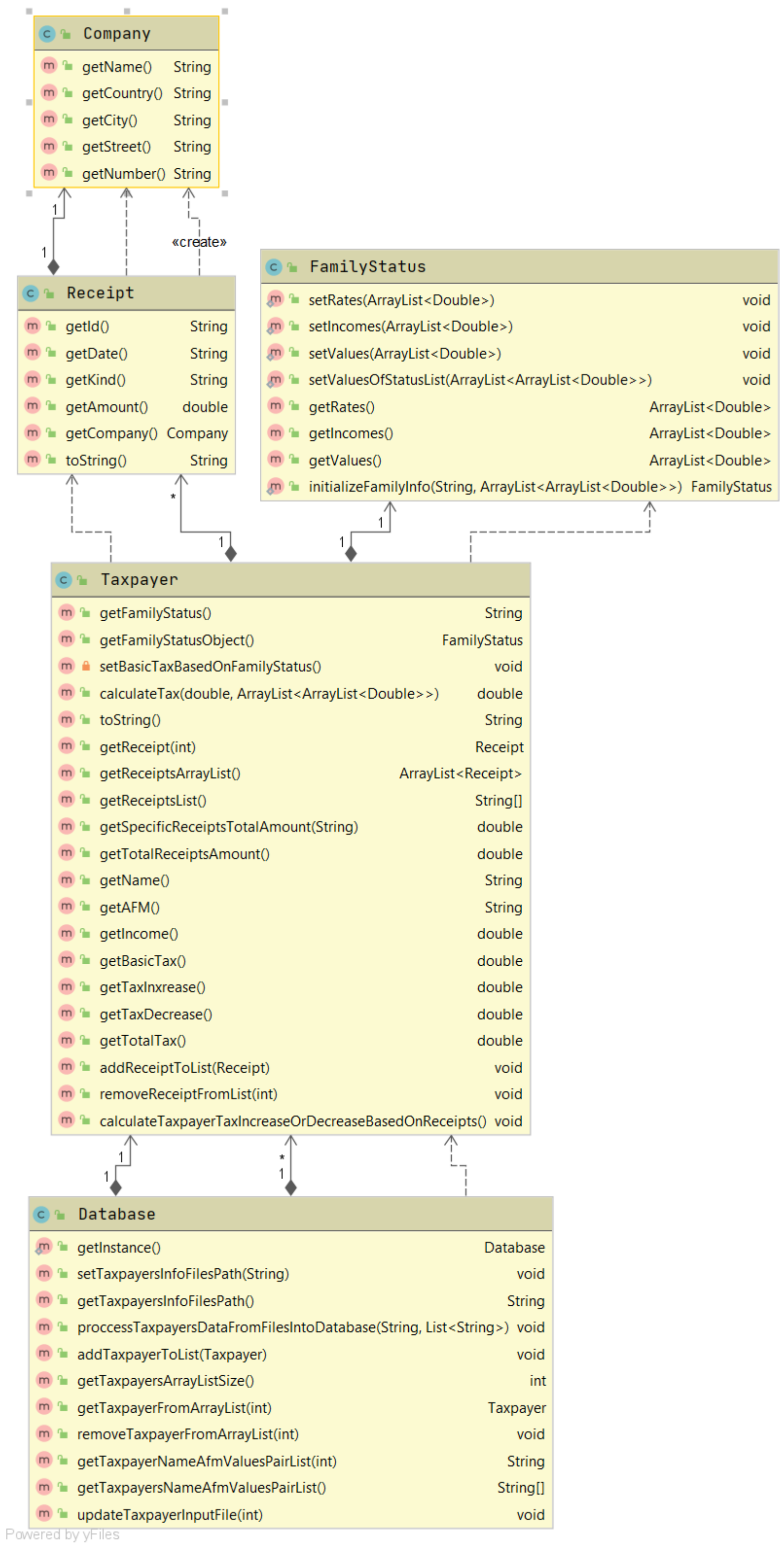
GENERAL UML FOR ALL CLASSES



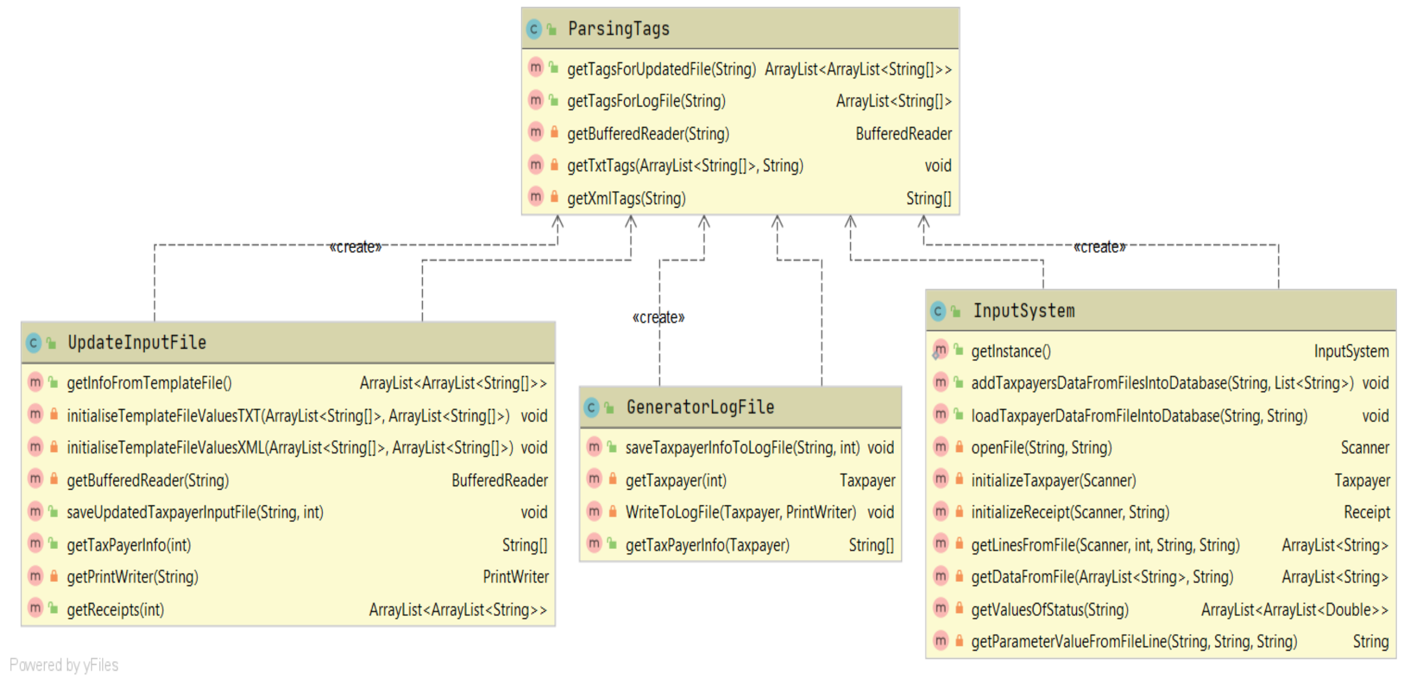
UML FOR CLASSES FROM GUI



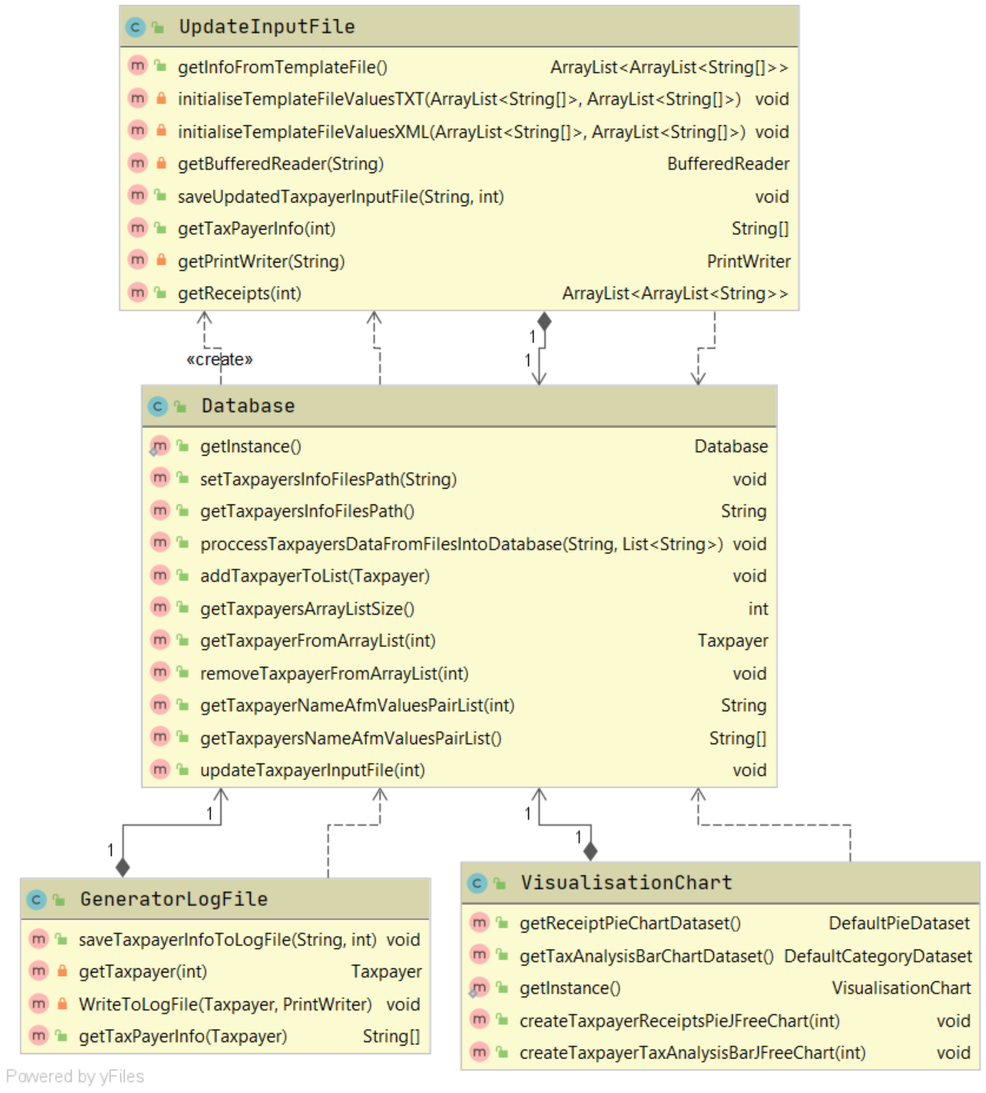
UML FOR CLASSES FROM DATAMANAGEPACKAGE


Company
getName() String
getCountry() String
getCity() String
getStreet() String
getNumber() String
«create»
1
1
Receipt
getId() String
getDate() String
getKind() String
getAmount() double
getCompany() Company
toString() String
*
1
FamilyStatus
setRates(ArrayList<Double>) void
setIncomes(ArrayList<Double>) void
setValues(ArrayList<Double>) void
setValuesOfStatusList(ArrayList<ArrayList<Double>>) void
getRates() ArrayList<Double>
getIncomes() ArrayList<Double>
getValues() ArrayList<Double>
initializeFamilyInfo(String, ArrayList<ArrayList<Double>>) FamilyStatus
1
1
Taxpayer
getFamilyStatus() String
getFamilyStatusObject() FamilyStatus
setBasicTaxBasedOnFamilyStatus() void
calculateTax(double, ArrayList<ArrayList<Double>>) double
toString() String
getReceipt(int) Receipt
getReceiptsArrayList() ArrayList<Receipt>
getReceiptsList() String[]
getSpecificReceiptsTotalAmount(String) double
getTotalReceiptsAmount() double
getName() String
getAFM() String
getIncome() double
getBasicTax() double
getTaxInxrease() double
getTaxDecrease() double
getTotalTax() double
addReceiptToList(Receipt) void
removeReceiptFromList(int) void
calculateTaxpayerTaxIncreaseOrDecreaseBasedOnReceipts() void
1
1
*
1
Database
getInstance() Database
setTaxpayersInfoFilesPath(String) void
getTaxpayersInfoFilesPath() String
proccessTaxpayersDataFromFilesIntoDatabase(String, List<String>) void
addTaxpayerToList(Taxpayer) void
getTaxpayersArrayListSize() int
getTaxpayerFromArrayList(int) Taxpayer
removeTaxpayerFromArrayList(int) void
getTaxpayerNameAfmValuesPairList(int) String
getTaxpayersNameAfmValuesPairList() String[]
updateTaxpayerInputFile(int) void
Powered by yFiles

UML FOR INPUT – OUTPUT CLASSES AND PARSE TAGS CLASS INTERACTION


ParsingTags
getTagsForUpdatedFile(String) ArrayList<ArrayList<String[]>>
getTagsForLogFile(String) ArrayList<String[]>
getBufferedReader(String) BufferedReader
getTxtTags(ArrayList<String[]>, String) void
getXmlTags(String) String[]
«create»
«create»
«create»
UpdateInputFile
getInfoFromTemplateFile() ArrayList<ArrayList<String[]>>
initialiseTemplateFileValuesTXT(ArrayList<String[]>, ArrayList<String[]>) void
initialiseTemplateFileValuesXML(ArrayList<String[]>, ArrayList<String[]>) void
getBufferedReader(String) BufferedReader
saveUpdatedTaxpayerInputFile(String, int) void
getTaxPayerInfo(int) String[]
getPrintWriter(String) PrintWriter
getReceipts(int) ArrayList<ArrayList<String>>
GeneratorLogFile
saveTaxpayerInfoToLogFile(String, int) void
getTaxpayer(int) Taxpayer
WriteToLogFile(Taxpayer, PrintWriter) void
getTaxPayerInfo(Taxpayer) String[]
InputSystem
getInstance() InputSystem
addTaxpayersDataFromFilesIntoDatabase(String, List<String>) void
loadTaxpayerDataFromFileIntoDatabase(String, String) void
openFile(String, String) Scanner
initializeTaxpayer(Scanner) Taxpayer
initializeReceipt(Scanner, String) Receipt
getLinesFromFile(Scanner, int, String, String) ArrayList<String>
getDataFromFile(ArrayList<String>, String) ArrayList<String>
getValuesOfStatus(String) ArrayList<ArrayList<Double>>
getParameterValueFromFileLine(String, String, String) String
Powered by yFiles

UML FOR OUTPUT CLASSES - DATABASE INTERACTION


UpdateInputFile
getInfoFromTemplateFile() ArrayList<ArrayList<String[]>>
initialiseTemplateFileValuesTXT(ArrayList<String[]>, ArrayList<String[]>) void
initialiseTemplateFileValuesXML(ArrayList<String[]>, ArrayList<String[]>) void
getBufferedReader(String) BufferedReader
saveUpdatedTaxpayerInputFile(String, int) void
getTaxPayerInfo(int) String[]
getPrintWriter(String) PrintWriter
getReceipts(int) ArrayList<ArrayList<String>>
«create»
1
1
Database
getInstance() Database
setTaxpayersInfoFilesPath(String) void
getTaxpayersInfoFilesPath() String
proccessTaxpayersDataFromFilesIntoDatabase(String, List<String>) void
addTaxpayerToList(Taxpayer) void
getTaxpayersArrayListSize() int
getTaxpayerFromArrayList(int) Taxpayer
removeTaxpayerFromArrayList(int) void
getTaxpayerNameAfmValuesPairList(int) String
getTaxpayersNameAfmValuesPairList() String[]
updateTaxpayerInputFile(int) void
1
1
1
1
GeneratorLogFile
saveTaxpayerInfoToLogFile(String, int) void
getTaxpayer(int) Taxpayer
WriteToLogFile(Taxpayer, PrintWriter) void
getTaxPayerInfo(Taxpayer) String[]
VisualisationChart
getReceiptPieChartDataset() DefaultPieDataset
getTaxAnalysisBarChartDataset() DefaultCategoryDataset
getInstance() VisualisationChart
createTaxpayerReceiptsPieJFreeChart(int) void
createTaxpayerTaxAnalysisBarJFreeChart(int) void
Powered by yFiles

Bellow we are going to address the issues that the previous code had and refactoring methods that we implemented in order to resolve them.

Issue 1: The Taxpayer class had a lot of code duplication.

1. Specifically, we observed that all the calculateTax* methods were very similar with only difference some constants which were depended on the family status of the taxpayer.

   Refactoring: to resolve this issue we created a new general parameterize method that implements the main algorithm of the tax calculation and takes as arguments the rates and the incomes of the specific taxpayer.

2. In addition, there was another main duplication about get*receiptsTotalAmount methods. There were many similar methods which were calculating the total amount of a specific kind receipts.

   Refactoring: to resolve this issue we created a new general parameterize method that takes as an argument the type of the receipts that we want to calculate.

Issue 2: The Taxpayer class suffers from primitive obsession.

The specific class contains many constants for each taxpayer. These constants relate to the income limits, the income tax rates and some values that been needed to calculate the tax of a taxpayer with a specific family status. Essentially the family status acts as a typecode for different kinds of taxpayers.

Refactoring: to resolve this problem we choose to replace the typecode (familystatus) with a new Class that it will manage the above values for every different familystatus. By this way, in order to determine the familystatus of the current taxpayer we use its objects instead of the type code values. Finally, we created four different static fields, for each familystatus, that we instantiate at the inputSystem and intialize its values from a property file that we created.

Issue 3: <u>The different subclasses of Receipt are trivial</u>

Receipt subclasses are very small, and they can be considered as lazy classes since they don't have any important responsibilities.

<u>Refactoring:</u> To resolve the above problem we deleted all these classes, and we moved their features to the main class Receipt.

Issue 4: <u>There are some classes that don't follow the object-oriented style</u>

The main issue here is that the following classes use static methods and static fields to communicate with each other or with other classes of the application. The classes that we are talking are the Database, the InputSystem and the OutputSystem.

<u>Refactoring:</u> We want to turn all the methods and fields into normal and not static. For these specific classes we need to have one object of every class in our application. For this reason, we decided to choose the Singlenton Pattern. By implementing this pattern, we ensure that every of the above classes will have only one instance, while it is providing a global access point to this instance.

Issue 4: <u>The InputSystem class has too many responsibilities</u>

The specific class contains the implementation of parsing two different file formats (XML and TXT).

<u>Refactoring 1:</u> To avoid having too many responsibilities in InputSystem we decided to split it in two subclasses which implement the two different formats. In order to achieve this, we used the Template Method. More specifically, in InputSystem we broke down the main format "algorithm" into a series of methods and we put a series of calls to these methods inside a single *template method* that contains the skeleton of that "algorithm". One of these methods is an abstract method that the children of the class implement the different code for two formats XML and TXT.

Refactoring 2: In addition, we noticed that the code could be minimized further. The sublclasses' code is similar so now we have a problem of duplication. For this reason, we managed to minimize the implementation to a single InputSystem class for parsing. For the tags we created two different files that contain the format of the different type of file (XML, TXT) and we derived a new class (ParsingTags) with only responsibility to keep these tags in structures. The new InputSystem Class calls the previous method passing as argument the appropriate file in order to receive the right tags.

Issue 5: The OutputSystem class has too many responsibilities

The main issue here is that this class handles too many operations, such as visualization of data, exporting new files and updating existed files.

Refactoring: In order to resolve the issue above we replaced OutputSystem Class with 3 different classes which are responsible for the three different operations that the previous class was implementing all together. In addition, we observed that there was some more duplication in the classes which were handling the export of new file and the update of an existed one. In other words, the tags that we used for the updated file and the new file were following a similar structure as the file of the InputSystem class. So, we concluded to use the class (ParsingTags) that we created to the previous issue in order to manage the tags of the new and the update file. For the updates operation we used the files that we created earlier (for the input tags structure) and for the export new file we created two new files (for txt and xml) with the new format.

IMPLEMENTATION

In the following pages we are going to give CRC cards for the classes of every package.

dataManagePackage

Class Name: Company	
Responsibilities	**Collaborations**
• This class is responsible of initializing and returning the Company information for every Receipt.	• Receipt

Class Name: Database	
Responsibilities	**Collaborations**
• This class interacts with almost all the classes of the project • It is responsible of managing new taxpayers and already existed taxpayers in the project. • Responsibilities: delete and add information of the taxpayer to the system and return them to other classes.	• Taxpayer • InsertNewReceiptJDialog • LoadedTaxpayersJDialog • TaxpayerLoadDataJDialog • TaxpayerReceiptsManagementJDialog • MainJFrameWindow • InputSystem • GeneratorLogFile • UpdateInputFile • VisualisationChart

Class Name: FamilyStatus	
Responsibilities	**Collaborations**
• This class has 2 responsibilities: ○ To initialize the family status of every taxpayer ○ To initialize important values to calculate taxpayer's tax	• Taxpayer • InputSystem

Class Name: Receipt	
Responsibilities	**Collaborations**
• This class is responsible of initializing and returning the Receipt's information.	• InputSystem • Taxpayer • InsertNewReceiptJDialog • UpdateInputFile • Company

Class Name: Taxpayer	
Responsibilities	**Collaborations**
• This class is responsible of initializing and returning the taxpayer’s information. • In addition, other responsibilities of this class are, the calculations of the taxpayer’s taxes.	• Receipt • FamilyStatus • InputSystem • GeneratorLogFile • VisualisationChart • Database

inputManagePackage

Class Name: InputSystem	
Responsibilities	**Collaborations**
• This class is responsible for loading a taxpayer’s data in database. • More specific this class update the input files which keep information for a taxpayer. • As a first step this class takes basic information about a taxpayer such as their name and in addition go on for their receipt condition. • For instance, if the user adds a receipt this class will activate updating the specific input file.	• ParsingTags • FamilyStatus • Receipt • Taxpayer • Database

outputManagePackage

Class Name: VisualizationChart	
Responsibilities	**Collaborations**
• This class has two responsibilities. ○ visualize tax condition of a taxpayer with a PieJFreeChart ○ visualize tax condition of a taxpayer with a BarJFreeChart	• Database • Taxpayer • LoadedTaxpayersJDialog

Class Name: UpdateInputFile	
Responsibilities	**Collaborations**
• The main responsibility for this class is to update the taxpayer's data. • For instance, it updates the receipts of the taxpayer, when the user deletes or insert a new receipt.	• Receipt • Database

Class Name: GeneratorLogFile	
Responsibilities	**Collaborations**
• Main responsibility for this class is to export information about taxpayers' purchases. • In other words, exports to a log file information about the receipts and tax fluctuation.	• ParsingTags • Taxpayer • Database • LoadedTaxpayersJDialog

Gui

Class Name: MainJFrameWindow	
Responsibilities	**Collaborations**
• This class provides a window with buttons that have two responsibilities. • The first button is responsible for loading a folder that contains files with taxpayers' data. • The second button is responsible for appearing a list of all taxpayers' files.	• LoadedTaxpayersJDialog • TaxpayerLoadDataJDialog • Database

• **Class Name: TaxpayerLoadDataJDialog**	
Responsibilities	**Collaborations**
• This class provides a window that shows the files of the folder that the user has chosen in the main window • Main responsibility of this class is to give the right to the user to select and load to the system the files that he prefers.	• MainJFrameWindow • Database

Class Name: InsertNewReceiptJDialog	
Responsibilities	**Collaborations**
• This class provides a window that shows a form in which the user can fill in all the information of a new receipt and add it to the system.	• Receipt • Database • TaxpayerReceiptsManagementJDialog

Class Name: LoadedTaxpayersJDialog	
Responsibilities	**Collaborations**
• This class provides a window that contains buttons that give the opportunity to the user to handle the main changes of the taxpayers' info • In other words, in this window the user can see the taxpayers' info and receipts, to delete a taxpayer and to visualize taxpayer's data in a pieChart or a BarChart • The two last buttons of this window have the operation of exporting data in a new file.	• MainJFrameWindow • TaxpayerReceiptsManagementJDialog • Database • GeneratorLogFile • VisualizationChart

Class Name: `TaxpayerReceiptsManagementJDialog`	
Responsibilities	**Collaborations**
• This class provides a window that contains the list with all the receipts of the selected taxpayer • In addition, it contains the three following buttons ○ Appearance specific receipt's information ○ Insert New Receipt ○ Delete Selected Receipt	• LoadedTaxpayersJDialog • Database • InsertNewReceiptJDialog

<u>ManagerTags</u>

Class Name: ParsingTags	
Responsibilities	**Collaborations**
• This class has two responsibilities: ○ It reads and keeps the tags of the initial files in structures from some format files that we created ○ It reads and keeps the tags of the export files in structures from some format files that we created	• InputSystem • GeneratorLogFile • UpdateInputFile

<u>Links:</u>

<u>Video(</u>*detailed explanation of the refactoring***):** https://github.com/NikosSpyropoulos/Refactor-Minnesota-Income-Tax-Calculation/blob/master/demo_Video.mp4

<u>Project Repository on GitHub:</u> https://github.com/NikosSpyropoulos/Refactor-Minnesota-Income-Tax-Calculation