<MINNESOTA INCOME TAX CALCULATION >

OVERALL REPORT

VERSION <1.0>

<Alexiou Alexandros 2929>

<email cs02929@uoi.gr>

<Krokos Athanasios 3012>

<email cs03012@uoi.gr>

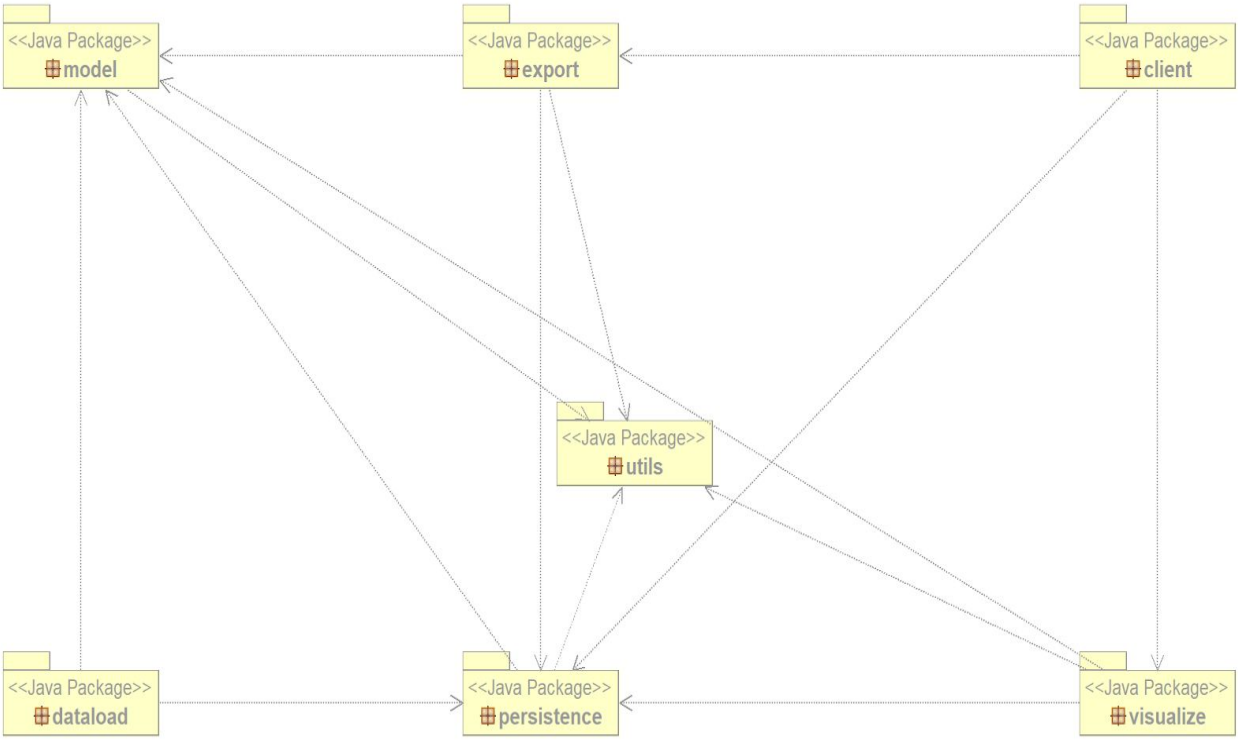
TABLE OF CONTENTS

INTRODUCTION

This project is a reengineer of a legacy Java application. It serves as an income tax calculator for the citizens of Minnesota, based on the tax calculation algorithms by the state of Minnesota, the receipts of each citizen, their yearly income and their family status. The necessary data for the calculation are given as .txt or .xml files for each citizen. Finally, the application produces output reports in .txt or .xml format and visualizes the data as bar or pie charts.

REFACTORED DESIGN

ARCHITECTURE



The refactored application's code is split into 7 packages.

1. model

   The package "model" includes all the data logic of the application.

2. export

   The package "export" contains the classes that are responsible for the output files.

3. dataload

   The package "dataload" contains the classes that are responsible for reading the input files (xml, txt).

4. client

   The package "client" contains the classes responsible for the graphical user interface of the application.

5. persistence

   The package "persistence" includes a simple database that stores the taxpayers and their data (receipts).

6. visualize

   The package "visualize" includes the functionality of the bar and pie chart creation.

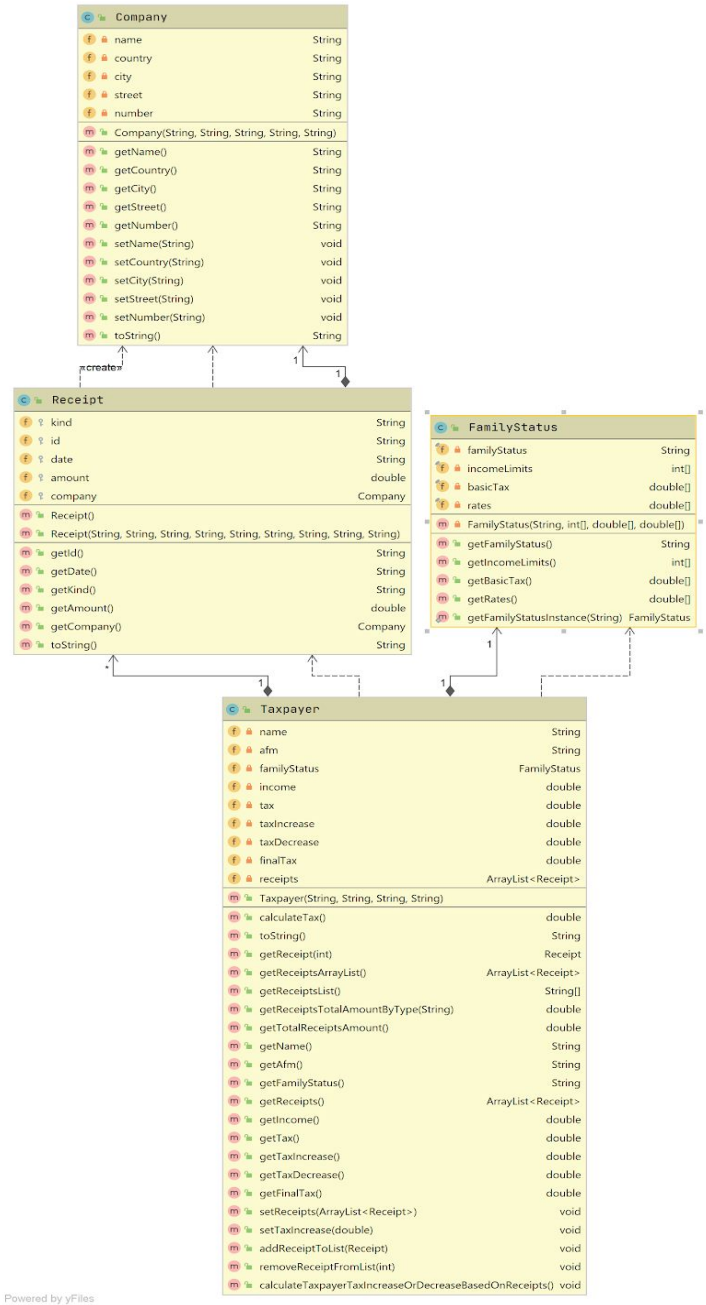
7. utils

   The package "utils" includes the application constants and error messages.

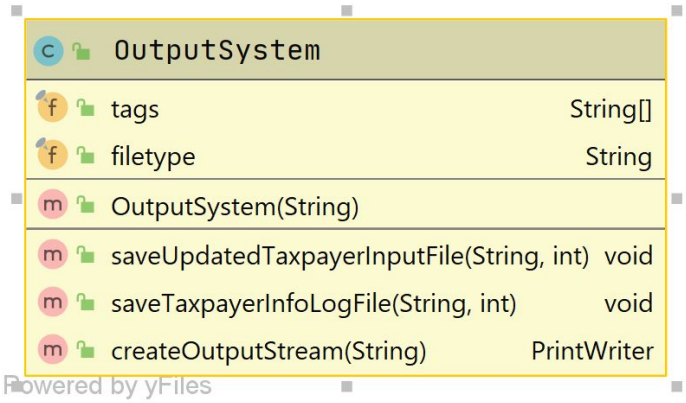
DETAILED DESIGN

UML Diagrams for each package

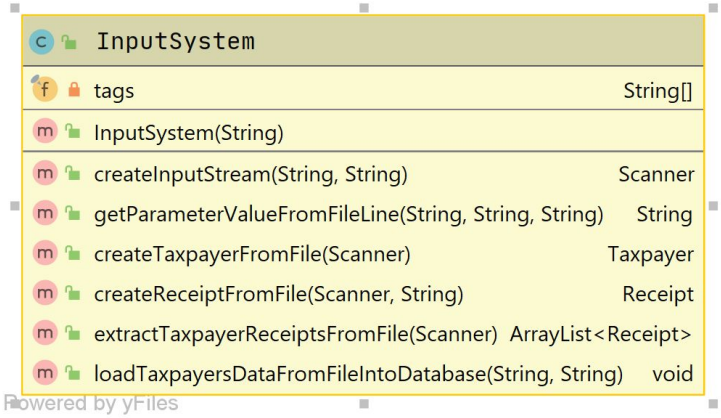
1. Package “model”



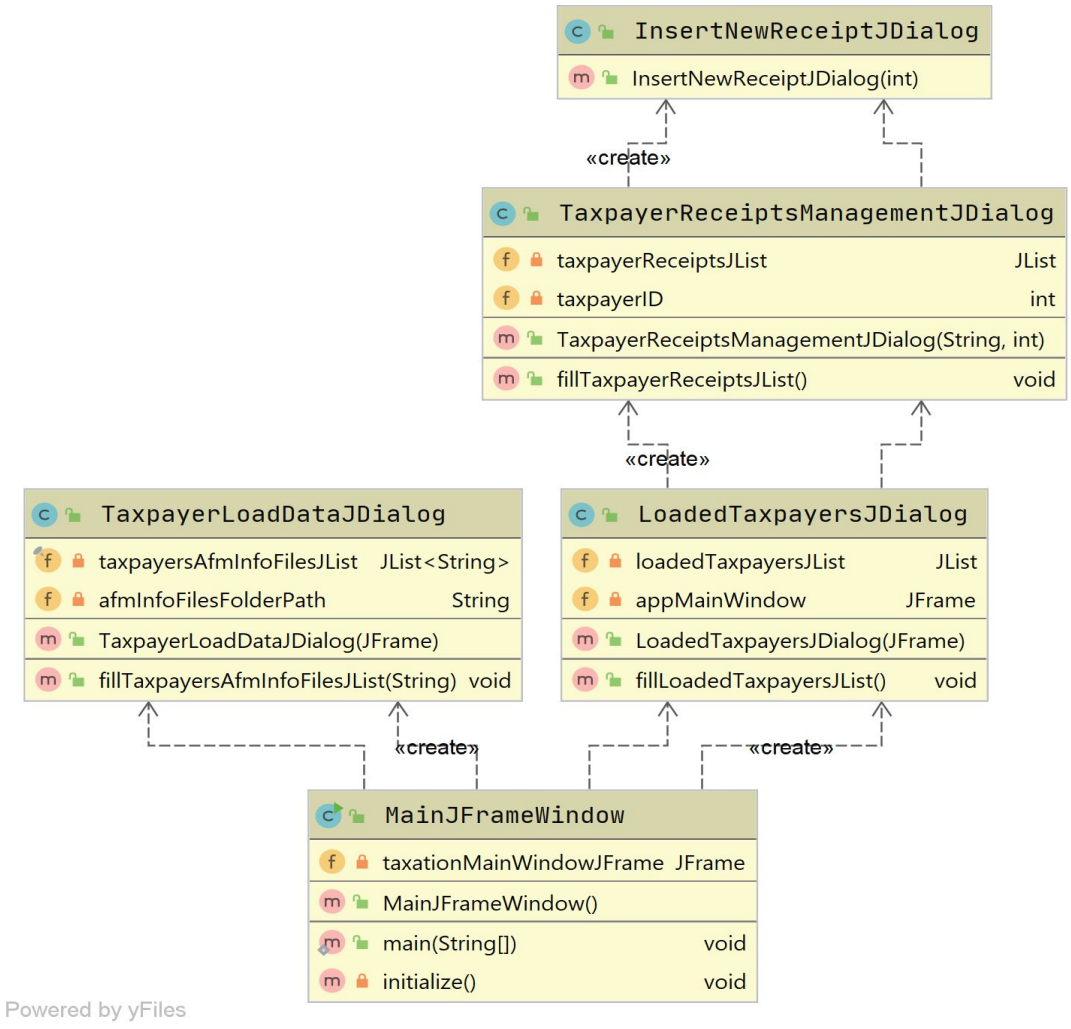
2. Package “export”



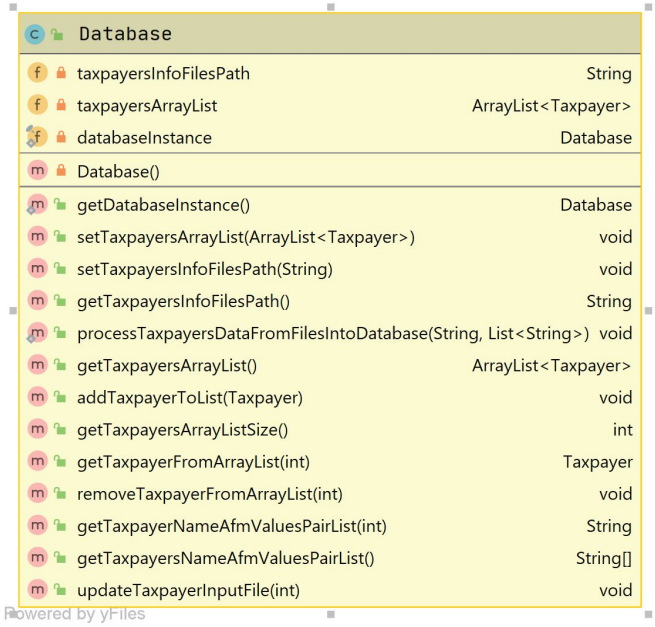
3. Package “dataload”



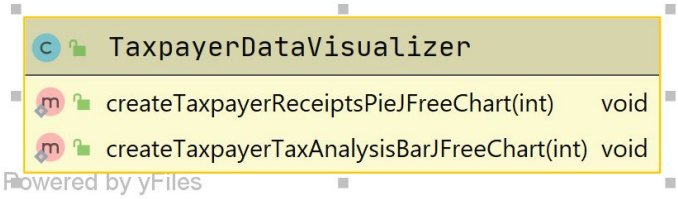
4. Package “client”



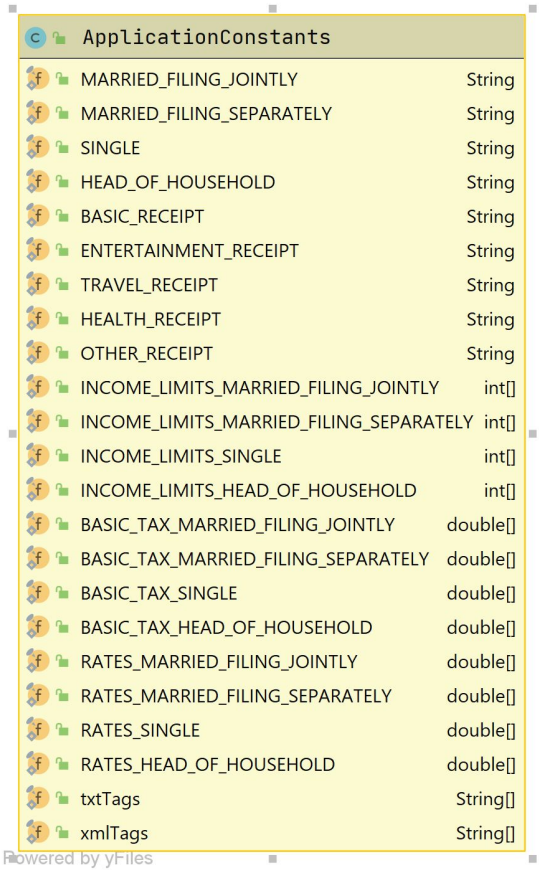
5. Package “persistence”

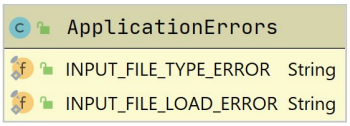


6. Package “visualize”



7. Package “utils”





PROBLEMS WITH THE OLD DESIGN

1. Code duplication

This problem appears in the class Taxpayer. The groups of methods named “calculateTaxFor..” and “get...ReceiptsTotalAmount” are similar, their only difference is in some constants.

In order to unify these methods, we create generic methods. These methods are now parameterized and combine the functionality of the former methods. Instead of 2 groups with similar methods we now have 2 parameterized methods that work generically.

2. Primitive obsession

This problem occurs from having constants as primitive types inside the classes. Primitive obsession appears in the Taxpayer class for the “family_status” field and in the Receipt class for the “type” field.

To fix this problem, we created a new class named “Application constants”. There, the constants are stored as string fields and are called as fields from the class when needed.

3. Lazy classes

The different subclasses of the class Receipt are redundant. They do not offer any additional functionality from the super class.

To solve this problem, we added a new String field named “type” to store each receipt’s type, so now the design is simplified.

4. Static methods and static fields

The Database, InputSystem and OutputSystem classes have static methods that operate on static fields.

To solve this problem, we made these methods non-static and refactored the classes using the Singleton pattern.

5. Too many responsibilities (InputSystem class)

The InputSystem class is used to parse two different input formats (XML and TXT). It had two methods (loadTaxpayersDataFromXmlFileIntoDatabase, loadTaxpayerDataFromTxtFileIntoDatabase) that were very similar. The only difference was with the different tags that were needed to parse the different file formats. To address this issue we created two arrays with the different tags, we derived one method for parsing, parameterizing it with the appropriate tags depending on the input file (txt or xml). With this refactoring the class is more concise with much less lines of code.

6.Too many responsibilities (OutputSystem class)

The OutputSystem class is responsible for generating output reports in two different formats (.txt, .xml) and for creating pie charts and bar charts.

To solve this problem we made the OutputSystem class abstract with the methods "saveUpdatedTaxpayerInputFile" and "saveTaxpayerInfoLogFile" declared as abstract. These methods are implemented in the classes "TxtFileOutput" and "XmlFileOutput" classes respectively. We parameterized the OutputSystem with the tags for the file types and we derived a single class for the file outputs.

The methods "createTaxpayerReceiptsPieJFreeChart" and "createTaxpayerTaxAnalysisBarJFreeChart" are moved to the package "visualize".

Now each class has only one responsibility.

IMPLEMENTATION

1. CRC cards for the package "client"

Class Name: MainJFrameWindow	
Responsibilities	**Collaborations**
▪ Creates the main window of the application. ▪ Creates 2 buttons for user interface.	▪ class TaxpayerLoadDataJDialog ▪ class LoadedTaxpayersJDialog

Class Name: TaxpayerLoadDataJDialog	
Responsibilities	**Collaborations**
▪ Opens a file chooser window . ▪ Gets the input file path as user input.	▪ class Database

Class Name:LoadedTaxpayersJDialog	
Responsibilities	**Collaborations**
▪ Opens a window with the list of the taxpayers in the database. ▪ Creates buttons for 1. Showing the information of a selected taxpayer. 2. Deleting the selected taxpayer. 3. Showing the list of receipts of the selected taxpayer. 4. Making a pie chart of the receipts of the selected taxpayer. 5. Making a bar chart of the selected taxpayers tax analysis.	▪ class OutputSystem ▪ class Database ▪ class TaxpayerDataVisualizer

Class Name: TaxpayerReceiptsManagementJDialog	
Responsibilities	**Collaborations**
▪ Opens a window with the list of the selected taxpayers receipts. ▪ Creates buttons for 1. Showing the information of the selected receipt. 2. Inserting a new receipt . 3. Deleting a receipt.	▪ class Database

Class Name: InsertNewReceiptJDialog	
Responsibilities	**Collaborations**
▪ Opens a window for entering the attributes of a blank receipt. ▪ Adds the new receipt to the list of the taxpayers receipts. ▪ Updates the taxpayer's input file, so that it includes the newly added receipt.	▪ class Database ▪ class Receipt

2. CRC cards for the package “dataload”

Class Name: InputSystem	
Responsibilities	**Collaborations**
・ Parses the input files and creates Taxpayer objects. ・ Parses the input files and creates Receipt objects. ・ Stores the objects in the database.	・ class Taxpayer ・ class Receipt ・ class Database ・ class AppicationErrors ・ class FileTypes

3. CRC cards for the package “export”

Class Name: OutputSystem	
Responsibilities	**Collaborations**
• Creates output reports in .txt or xml format . • Updates the taxpayer’s input file, as it changes from the application.	• class Taxpayer • class Receipt • class Database • class ApplicationConstants • class FileTypes

4. CRC cards for the package “model”

Class Name: Company	
Responsibilities	**Collaborations**
• Defines the attributes of the “Company” objects.	• class Receipt

• Accesses and mutates the fields of the "Company" objects.	

Class Name: FamilyStatus	
Responsibilities	**Collaborations**
• Defines the attributes of the "FamilyStatus" objects. • Accesses the fields of the "FamilyStatus" objects.	• class ApplicationConstants • class Taxpayer

Class Name: Receipt	
Responsibilities	**Collaborations**
• Defines the attributes of the "Receipt" objects.	• class Company • class Taxpayer

▪ Accesses the fields of the “Receipt” objects.	▪ class ApplicationConstants

Class Name: Taxpayer	
Responsibilities	**Collaborations**
▪ Defines the attributes of the “TaxPayer” objects. ▪ Accesses the fields of the “Taxpayer” objects.	▪ class FamilyStatus ▪ class Taxpayer ▪ class ApplicationConstants

5.CRC cards for the package “persistence”

Class Name: Database	
Responsibilities	**Collaborations**
▪ Adds taxpayers to the database. ▪ Deletes taxpayers from the database. ▪ Keeps the input files updated on any changes to the database.	▪ class OutputSystem ▪ class Taxpayer

6.CRC cards for the package “utils”

Class Name: ApplicationConstants	
Responsibilities	**Collaborations**
▪ Keeps the constants that are used in the application.	▪ class Receipt ▪ class Taxpayer ▪ class Database ▪ class FamilyStatus

Class Name: Application Errors	
Responsibilities	**Collaborations**
▪ Keeps the application errors that are used in the application.	▪ class InputSystem

Class Name: FileTypes	
Responsibilities	**Collaborations**
▪ Keeps the file type extensions .	▪ class OutputSystem ▪ class InputSystem

MAVEN

We used the maven project manager to handle the dependencies, the build and the testing process. We included the maven executable via the maven wrapper such that no installation of maven is required to build, test and run the project.

Run on Windows

Execute **mvnw clean install** to build and test the project. The jar will be in the target folder.

Run on posix

Execute **./mvnw clean install** to build and test the project. The jar will also be in the target folder.