INCOME TAX CALCULATOR

OVERALL REPORT

VERSION <1.0>

SIDIROPOULOS PANAGIOTIS 4489

TABLE OF CONTENTS

INTRODUCTION

The objective of this project is to re-engineer a Java application that calculates income tax for citizens of Minnesota state. The tax calculation takes into account the citizen's marital status, income, and expenses, as evidenced by a set of receipts submitted with the income. The legacy application takes txt or xml files as input that contain the necessary data for each citizen. The tax calculation is based on a complex algorithm provided by the state. The application also generates graphical representations of the data in the form of bar and pie charts, and produces output reports in txt or xml format.

The objectives of this phase of the project are to simplify the design to make it more extensible and maintainable, and to redesign the graphical interface to enhance its user-friendliness.

REFACTORED DESIGN

USE CASES

UC1: LoadTaxPayerFromFile	
Use case ID	UC1
Actors	The user of the application
Pre conditions	1. The user has started the application
Main flow of events	1. The use case starts when the user selects "load taxpayer" option from the application 2. The system displays the file explorer 3. The user browses the file explorer and selects the appropriate file 4. The user confirms the action 5. The system loads the taxpayer's information from the selected file
Post conditions	1. The tax information of the taxpayer has been loaded to the system

Alternative flow 1	1. A taxpayer already loaded exception is raised by the system 2. The application indicates that the taxpayer from the selected file is already loaded
Post conditions	-
Alternative flow 2	1. At any time the user may leave load taxpayer screen
Post conditions	-

UC2: DisplayTaxpayerInfo	
Use case ID	UC2
Actors	The user of the application
Pre conditions	The system has at least one taxpayer loaded
Main flow of events	1. The use case starts when the user selects a taxpayer from the list. 2. The system selects the taxpayer. 3. The user selects "view taxpayer information". 4. The system displays the taxpayer's information in a new screen
Post conditions	1. The selected taxpayer's information screen is shown to the user
Alternative flow 1	1. At any time the user may leave taxpayer info screen
Post conditions	1. -

UC3: AddReceiptToTaxpayer	
Use case ID	UC3
Actors	The user of the application
Pre conditions	The user is viewing the taxpayer info screen
Main flow of events	1. The use case begins when the user selects "Add Receipt." 2. The system displays a receipt form. 3. The user enters the receipt information into the form. 4. The user confirms the addition of the receipt. 5. The system adds the receipt to the taxpayer's list of receipts and updates the taxpayer info files.
Post conditions	1. The receipt has been added to the taxpayer, and the taxpayer info files have been updated
Alternative flow 1	1. The system raises a "wrong receipt input field" exception. 2. The application notifies the user that wrong information was entered into a field of the receipt form. 3. The application displays the input rules of the receipt form.
Post conditions	1. The use case continues at step 3 of the main flow
Alternative flow 2	1. The user may leave the receipt form at any time.
Post conditions	1. The addition of the receipt is canceled, and no receipt is added to the taxpayer

UC4: DeleteReceiptOfTaxpayer	
Use case ID	UC4

Actors	The user of the application
Pre conditions	The user views the taxpayer info screen
Main flow of events	1. The use case starts when the user selects a receipt and clicks on the "delete receipt" button 2. The system displays a confirmation message to confirm the deletion of the receipt 3. The user confirms the action 4. The system deletes the selected receipt and updates the taxpayer's information files
Post conditions	1. The selected receipt has been successfully deleted and the taxpayer's information files have been updated
Alternative flow 1	1. At any time the user may close the confirmation screen
Post conditions	1. The deletion is cancelled, and no receipt is deleted from the taxpayer's information files

UC5: DisplayTaxCharts	
Use case ID	UC5
Actors	The user of the application
Pre conditions	The user views the taxpayer info screen
Main flow of events	1. The use case starts when the user selects the “view reports” button 2. The system calculates the necessary data for the charts 3. The system displays charts showing information about receipts and taxes

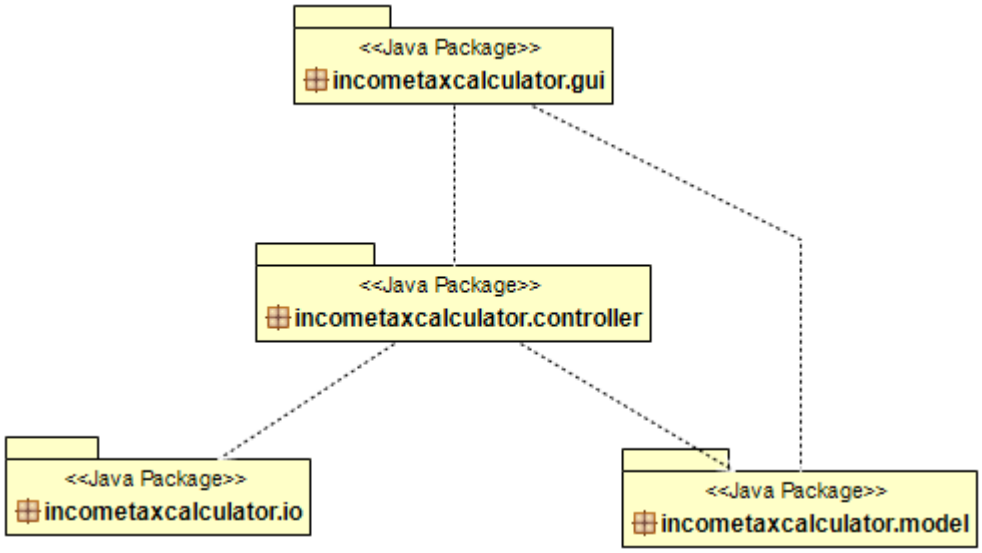
Post conditions	1. The reports are visible to the user

UC6: StoreTaxpayerLog	
Use case ID	UC6
Actors	The user of the application
Pre conditions	The user views the taxpayer info screen
Main flow of events	1. The use case starts when the user selects “save log” button 2. The system displays the file explorer 3. The user selects the directory where they want to save the log file 4. The system displays a message requesting the user to choose the file format from a list 5. The system saves the taxpayers info as a file with the specified format to the specified location 6. The system displays a confirmation message indicating that the log file has been saved
Post conditions	1. The taxpayers log is stored to the specified location with the specified format
Alternative flow 1	1. The user closes the file explorer
Post conditions	1. The log file is not saved
Alternative flow 2	2. The user closes the file format selection window
Post conditions	1. The log file is not saved

UC7: RemoveTaxapayer	
Use case ID	UC7
Actors	The user of the application
Pre conditions	1. Taxpayers are loaded into the system
Main flow of events	1. The use case starts when the user selects a taxpayer from the list and clicks on the "Remove taxpayer" button 2. The system displays a confirmation message asking the user if they want to proceed with the deletion 3. The user confirms their action 4. The system removes the selected taxpayer from the loaded taxpayer list
Post conditions	1. The selected taxpayer is removed from the loaded taxpayer list

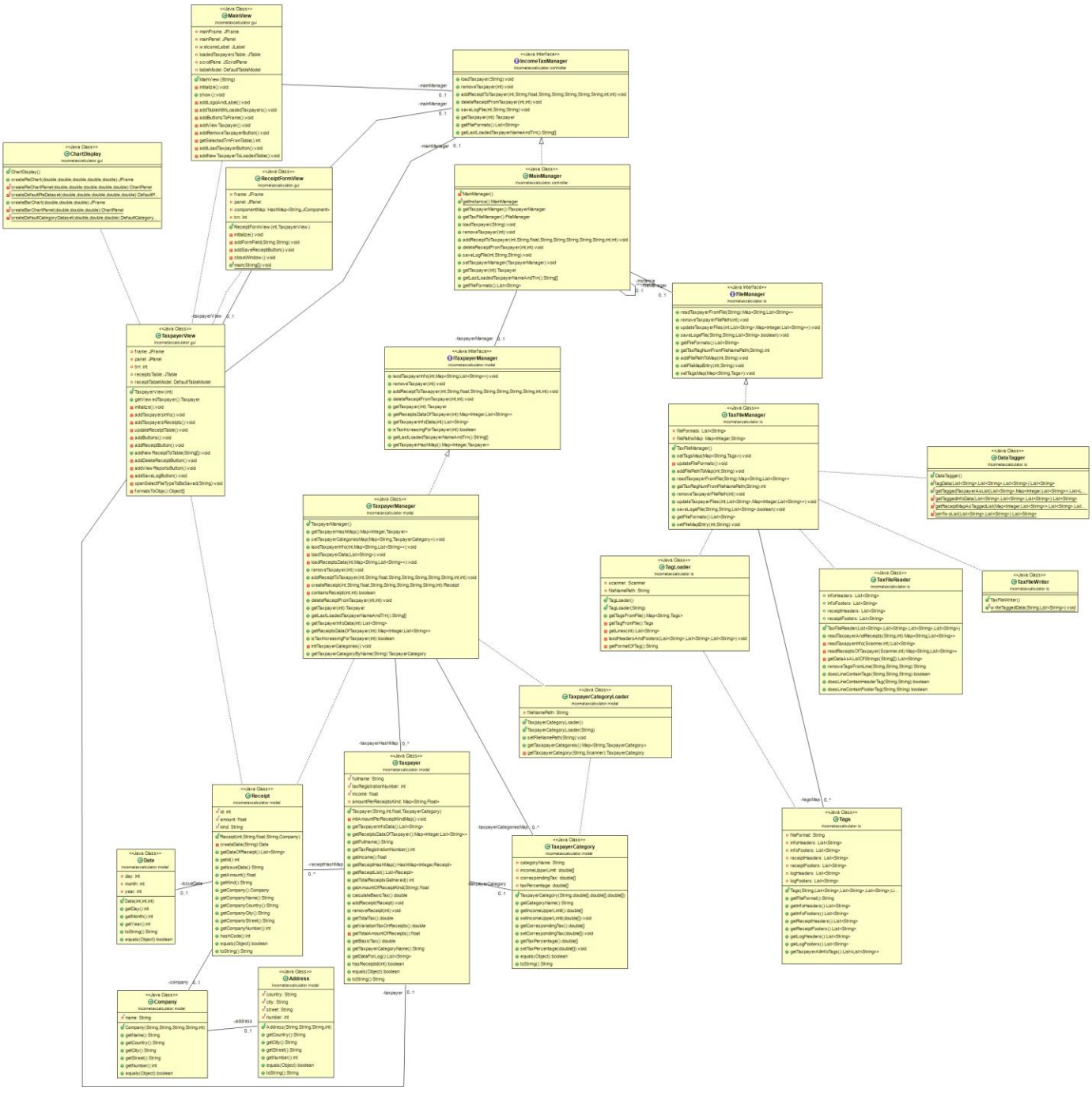
ARCHITECTURE

PACKAGE UML DIAGRAM:

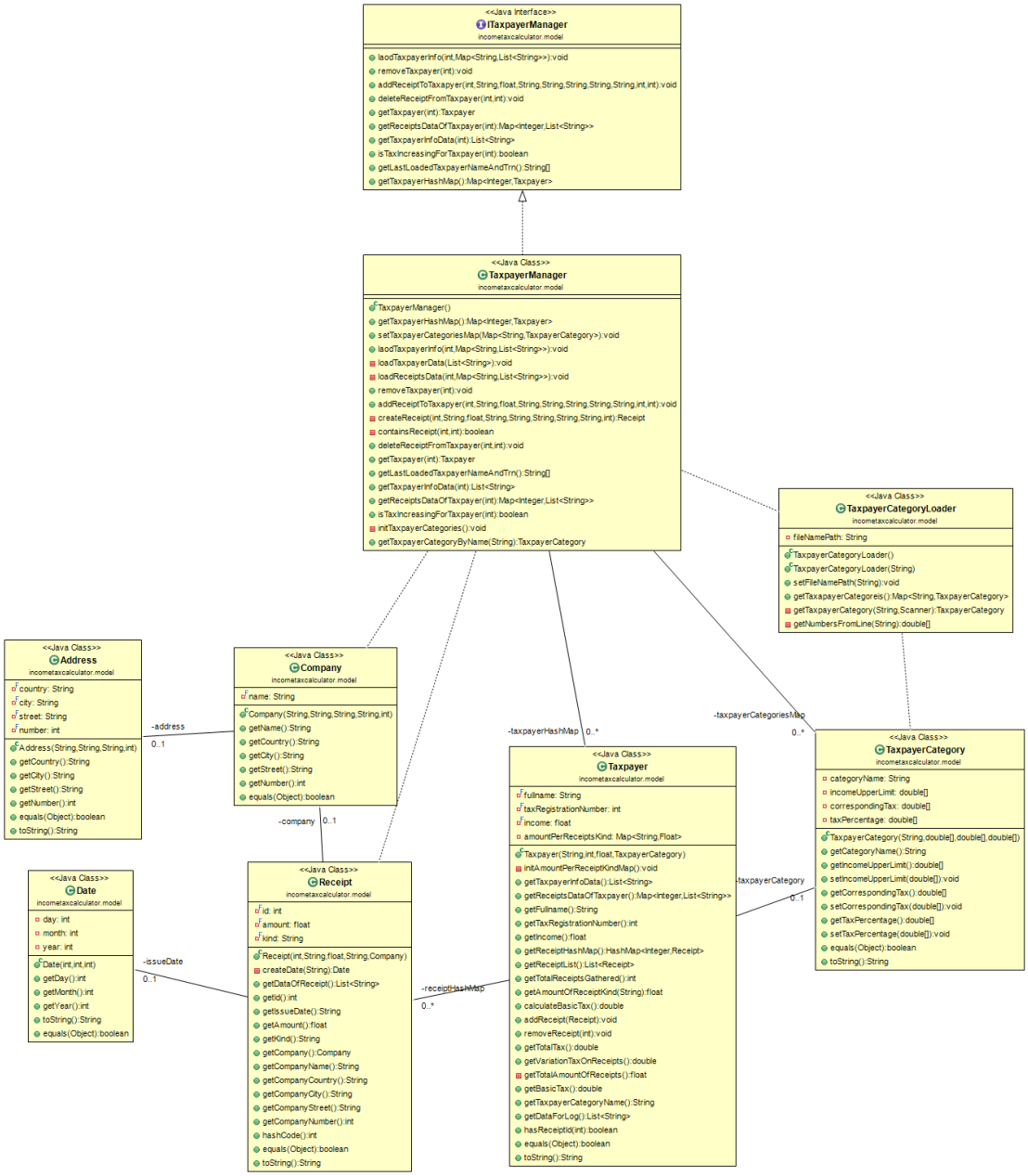


DETAILED DESIGN

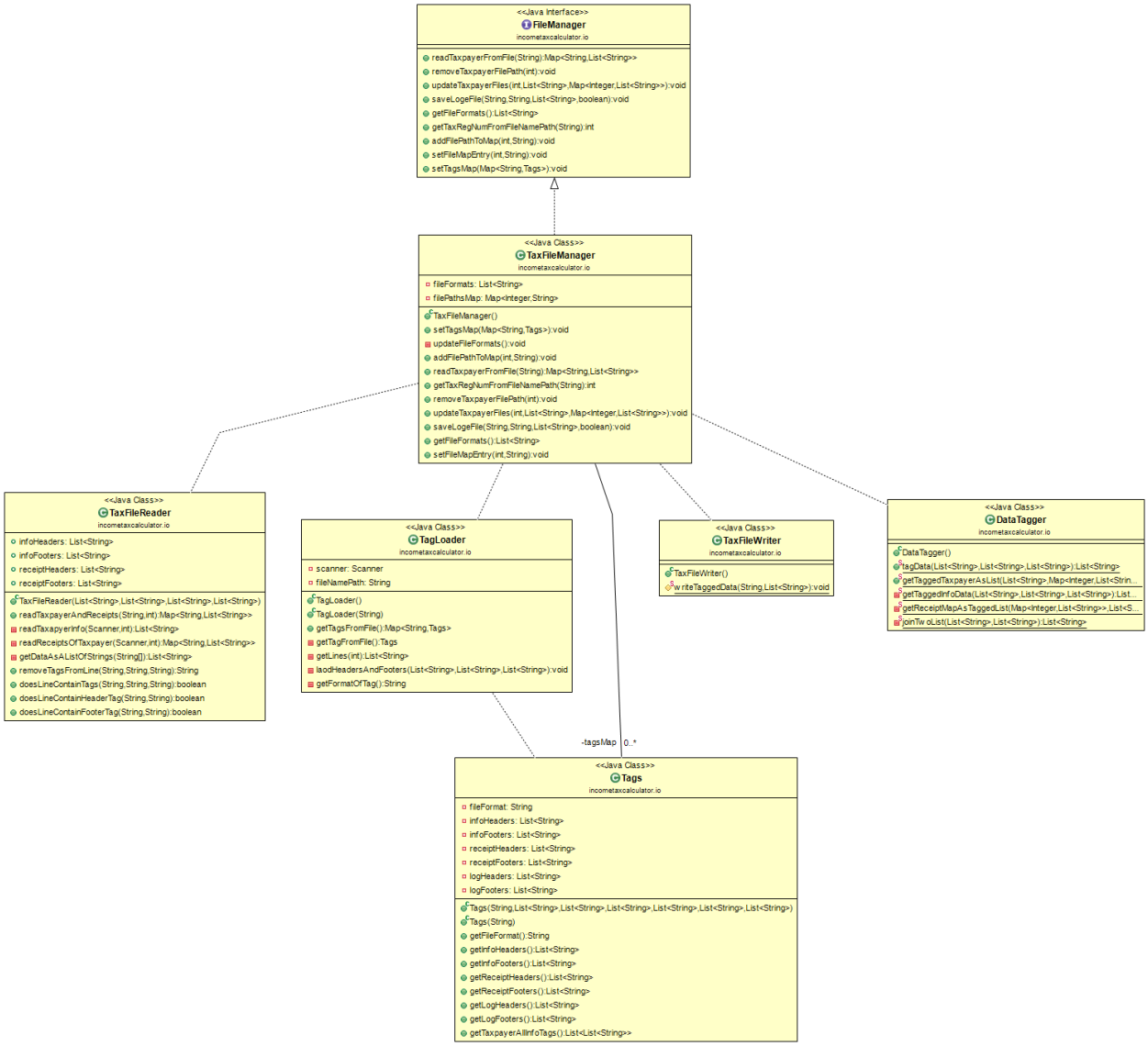
Project UML Class Diagram:



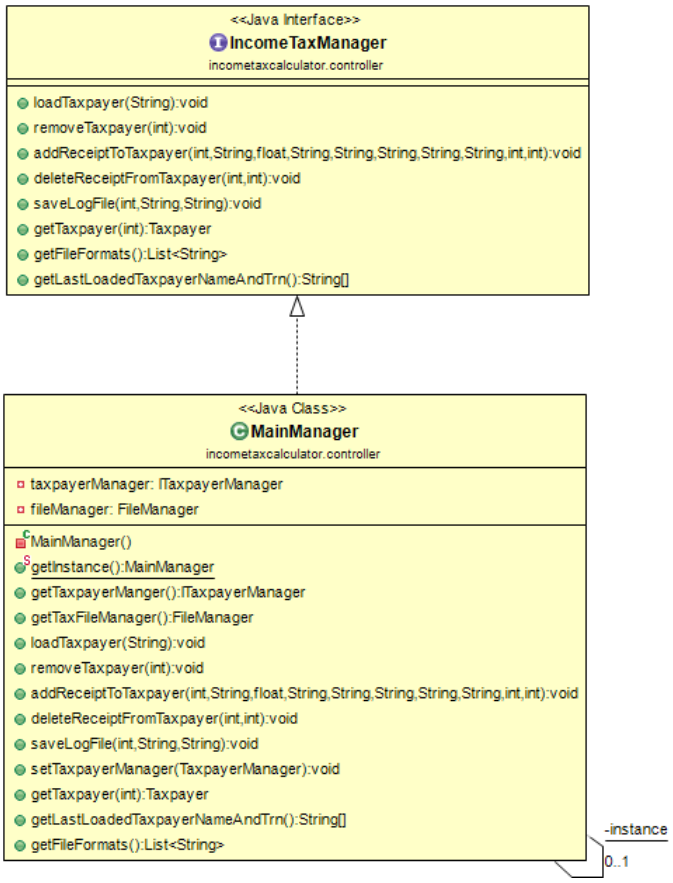
MODEL UML DIAGRAM:



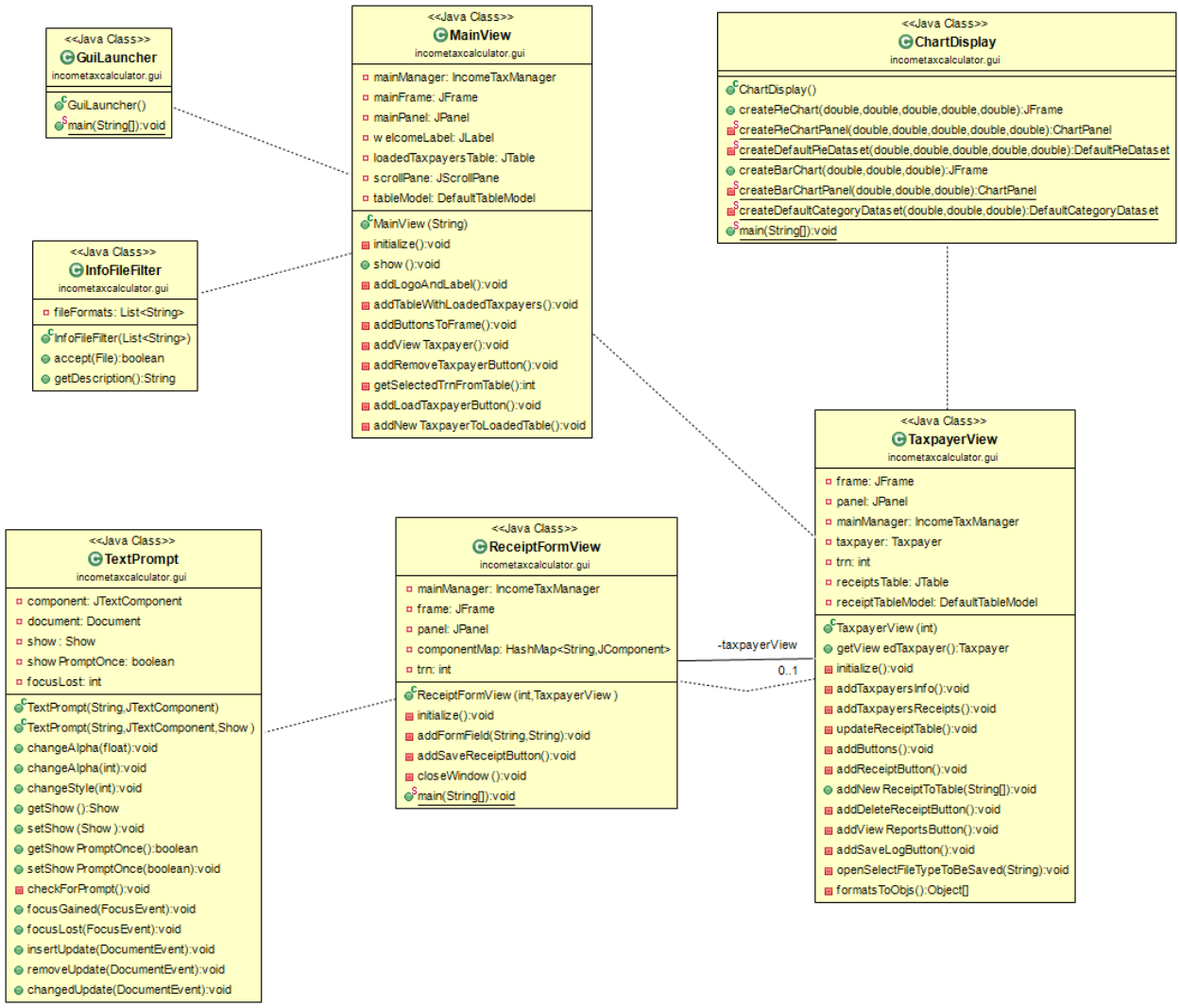
IO UML CLASS DIAGRAM:



CONTROLLER UML CLASS DIAGRAM:



GUI UML CLASS DIAGRAM:



SUMMARY OF CHANGES

1. The responsibilities of the TaxpayerManager have been separated into two subsystems: the TaxFileManager class, which is responsible for reading and writing data to and from files, and a separate TaxpayerManager class, which manages taxpayers. In addition, the MainManager class was implemented as a facade, which orchestrates the use of these two subsystems in implementing use cases.

2. To eliminate the need for Taxpayer subclasses, we utilized the "taxpayerProperties.txt" file, located in the "\resources" directory, to load constants for the Taxpayer class.

3. To eliminate the hierarchy of reader and writer classes, increase extensibility, and ensure that each field has the correct tags, we utilized the "tagsProperties.txt" file, located in the "\resources" directory, to load tags for different file types..

4. Addition of acceptance tests.

5. Changed the user interface to make it easier to use by providing multiple feedback messages and to enable the user to save and load taxpayer files from any file explorer folder on their computer and in any of the file formats that contained in the “tagsProperties.txt” file, located in the "\resources" directory,

6. We have made changes to class and package names, as well as reassigned classes to different packages.

CHANGES IN DETAIL:

LARGE CLASS TAXPAYERMANAGER

We have divided the TaxpayerManager into two separate classes. The first class, named TaxpayerManager, is responsible solely for managing taxpayers. The second class, named TaxFileManager, is responsible for handling files, including reading taxpayer information from a file and writing log and info files. As a result, the TaxpayerManager class has been modified to load the taxpayer data that was read into taxpayer objects. Adding receipts to a specific taxpayer is accomplished through a method within the taxpayer class..

TAXPAYER CHANGES

Initially, we utilized arrays in the base class of Taxpayer to hold constants that varied based on the taxpayer's type. However, as a result of this modification, the subclasses contained only constants and lacked behavior, so we decided to provide users with the ability to load class constants dynamically via a file named "taxpayerProperties.txt". We added a TaxpayerCategory class, which includes the taxpayer category's name and an array of constants for that category. The TaxpayerCategory class replaced the string status field of the Taxpayer, allowing us to calculate taxes based on the taxpayer's category. This approach makes it easy for users to add new taxpayer types. Moreover, the complex code that previously utilized chained if-else statements was replaced with a for loop and an internal check based on data that we stored in arrays.

DUPLICATE CODE IN INFO & LOG WRITER CLASSES:

Initially, we used separate template methods and factories to reduce duplicate code in these classes and to isolate the creation of their subclasses. However, as a result, the subclasses based on the file type contained only constants, which were the corresponding tags. Conceptually, we felt that the writer subclasses should not contain only constants, so we decided to give users the ability to dynamically load tags from a file (tagsProperties.txt) when starting the application. This approach allowed us to avoid large class hierarchies and to leave only one TaxFileWriter class that writes data to files. We utilized the DataTagger utility class to add tags to the data, and then used the TaxFileWriter to write the tagged data to the files needed for INFO and LOG, and any other types of files that may result from future expansion.

DUPLICATE CODE IN FILEREADER CLASSES

We worked similarly as for the Writers, with the difference that the remaining reader class (TaxFileReader) contains the methods responsible for removing the tags present in the Info files.

OTHER CHANGES:

We approached the Readers in a similar way as the Writers, with the exception that the remaining reader class (TaxFileReader) contains methods responsible for removing tags present in Info files.

CLASSES RESPONSIBILITIES AND COLLABORATIONS (CRC CARDS)

Class Name: MainManager	
Responsibilities	**Collaborations**
▪ Delegates user actions from gui to the respective subsystem	▪ TaxFile manager ▪ TaxpayerManger ▪ GUI Classes

Class Name: TaxpayerManger	
Responsibilities	**Collaborations**
▪ Loads taxpayer objects and adds them to loaded taxpayers ▪ Adds receipts to a specific taxpayer ▪ Deletes receipt of a taxpayer ▪ Removes taxpayers objects from loaded taxpayers ▪ Loads taxpayer categories from file	▪ Taxpayer ▪ Receipt ▪ TaxpayerCategoryLoader ▪ TaxpayerCategory ▪ Company

Class Name: Taxpayer	
Responsibilities	**Collaborations**
▪ Holds data of taxpayer ▪ Calculates the taxes of taxpayer ▪ Adds receipts from taxpayer ▪ Deletes receipt from taxpayer	▪ TaxpayerManger ▪ Receipt ▪ TaxpayerCategory ▪ TaxpayerView

Class Name: TaxpayerCategoyLoader	
Responsibilities	**Collaborations**
▪ Loads TaxpayerCategory objects from a file	▪ TaxpayerCategory ▪ TaxpayerManager

Class Name: TaxpayerCategory	
Responsibilities	**Collaborations**
▪ Holds data for the tax calculation of a taxpayer category	▪ Taxpayer ▪ TaxpayerManager

Class Name: Date	
Responsibilities	**Collaborations**
▪ Holds data of the issue date of a Receipt	▪ Receipt

Class Name: Company	
Responsibilities	**Collaborations**
▪ Holds data for the company issued a receipt	▪ Receipt

Class Name: Address	
Responsibilities	**Collaborations**
▪ Holds data of the address of the company	▪ Company

Class Name: Receipt	
Responsibilities	**Collaborations**

▪ Holds the data of a receipt	▪ Taxpayer ▪ Company ▪ Date ▪ TaxpayerManager ▪ TaxpayerView

Class Name: TaxFileManager	
Responsibilities	**Collaborations**
▪ Manages objects responsibles for: o Loading tags from file o Reading taxpayer info file o Updating taxpayer info files o Saving log file	▪ MainManager ▪ TagLoader ▪ TaxFileReader ▪ DataTagger ▪ TaxFileWriter

Class Name: TagLoader	
Responsibilities	**Collaborations**

▪ Loads tags from file	▪ Tag ▪ TaxFileManager

Class Name: TaxFileReader	
Responsibilities	**Collaborations**
▪ Reads taxpayer info file	▪ TaxFileManager

Class Name: TaxFileWriter	
Responsibilities	**Collaborations**
▪ Writes data to file	▪ TaxFileManager

Class Name: DataTagger	
Responsibilities	**Collaborations**
▪ Adds tags to data and returns tagged data	▪ TaxFileManager

Class Name: MainView

Responsibilities	Collaborations
▪ Displays main windows of the app containing the	▪ MainManager ▪ TaxpayerView

Class Name: TaxpayerView	
Responsibilities	**Collaborations**
▪ Displays taxpayer info and receipts	▪ MainView ▪ TaxpayerView ▪ ReceiptFormView ▪ Taxpayer ▪ Receipt

Class Name: ChartDisplay	
Responsibilities	**Collaborations**
▪ Creates and displays the charts	▪ TaxpayerView

Class Name: ReceiptFormView	
Responsibilities	**Collaborations**
▪ Displays a receipt form used for the addition of new receipt	▪ MainManager ▪ TaxpayerView