MINNESOTA INCOMA TAX CALCULATOR

OVERALL REPORT

VERSION <1.0>

Σκαρλάτου Δανάη, AM 2908

TABLE OF CONTENTS

INTRODUCTION

The objective of this educational project is to refactor the original Minnesota Income Tax Calculator and to improve the Graphical Interface.

REFACTORED DESIGN

USE CASES

<Use Case 1: Load taxpayer>

Use case ID	UC1
Actors	User
Pre conditions	The application is running
Main flow of events	1. The use case starts when the user clicks the “LOAD TAXPAYER” button 2. The file chooser dialog appears 3. The user browses for the file they want 4. The user confirms their selection by clicking the desired file and then “open” 4.1 If the Tax Registration Number of the chosen file is not displayed on the list, it is added to the list 4.2 If the Tax Registration Number of the chosen file is already displayed on the list, a message pops up to inform the user that the taxpayer is already loaded
Post conditions	The list contains the Tax Registration Number of the chosen file

<Use Case 2: Select Taxpayer>

Use case ID	UC2
Actors	User

Pre conditions	The application is running and the list of taxpayers is not empty
Main flow of events	1. The use case starts when the user clicks (selects) a loaded Tax Registration Number 2. The selected Tax Registration Number is highlighted 3. The user clicks the "SELECT TAXPAYER" button or double clicks the Tax Registration Number 4. A new window opens and displays the selected Taxpayer's information
Post conditions	Selected taxpayer window is open

<Use Case 3: Add Receipt>

Use case ID	UC3
Actors	User
Pre conditions	The application is running and the selected Taxpayer's information is displayed on a new screen
Main flow of events	1. The use case starts when the user clicks the "ADD RECEIPT" button 2. A dialog opens 3. The user fills the form of the dialog 3.1. If the user clicks ok, the form is checked 3.1.1. If the form passes the check, the new receipt is added to the list and the displayed information is updated 3.1.2. If the form fails the check, the user is notified of the problem with a dialog 3.2. If the user clicks cancel, the user returns to the previous screen
Post conditions	The selected taxpayer's information is displayed correctly

<Use Case 4: Delete Receipt>

Use case ID	UC4
Actors	User
Pre conditions	The application is running and the selected Taxpayer's information is displayed on a new screen
Main flow of events	1. The use case starts when the user clicks on a receipt on the list and then clicks the "DELETE RECEIPT" button 2. A dialog opens and asks the user if they are certain for the deletion 2.1. If the user clicks yes 2.1.1. The selected receipt is deleted from the table 2.1.2. The associated file is updated 2.1.3. The displayed information is updated 2.2. If the user clicks no 2.2.1. The user returns to the Selected Taxpayer's Information
Post conditions	The selected taxpayer's information is displayed correctly

<Use Case 5: Show Charts>

Use case ID	UC5
Actors	User
Pre conditions	The application is running and the selected Taxpayer's information is displayed on a new screen
Main flow of events	1. The user clicks the "VIEW CHARTS" button 2. A new window opens and displays a bar chart with the basic tax, total tax, and the tax variation

	3. A new window opens and displays a pie chart with the percentage of the total amount of each kind of receipt
Post conditions	Both of the charts are displayed correctly

<Use Case 6: Save Data>

Use case ID	UC6
Actors	User
Pre conditions	The application is running and the selected Taxpayer's information is displayed on a new screen
Main flow of events	1. The user clicks the "SAVE DATA" button 2. A dialog opens and asks the user for the file format 2.1. If the user clicks ok 2.1.1. If a _LOG file already exists, it is updated 2.1.2. If a _LOG file does not exist, it is created 2.2. If the user clicks cancel the dialog closes
Post conditions	The selected taxpayer's information is displayed correctly

<Use Case 7: Delete Taxpayer>

Use case ID	UC7
Actors	User
Pre conditions	The application is running and the list of taxpayers is not empty

Main flow of events	1. The use case starts when the user clicks (selects) a loaded Tax Registration Number 2. The selected Tax Registration Number is highlighted 3. The user clicks the “DELETE TAXPAYER” button 4. A dialog opens and asks the confirmation of the deletion 4.1. If the user clicks yes, the selected Tax Registration Number is deleted from the list, but the associated file is not deleted 4.2. If the user clicks no, the user returns to the original screen
Post conditions	The user returns to the main screen successfully

<EXTRA USE CASES>

<Use Case 8: Load All>

Use case ID	UC8
Actors	User
Pre conditions	The application is running
Main flow of events	1. The use case starts when the user clicks the “LOAD ALL” button 2. The list displays all the Tax Registration Numbers of the current directory
Post conditions	The list displays all the taxpayers

<Use Case 9: Create Taxpayer>

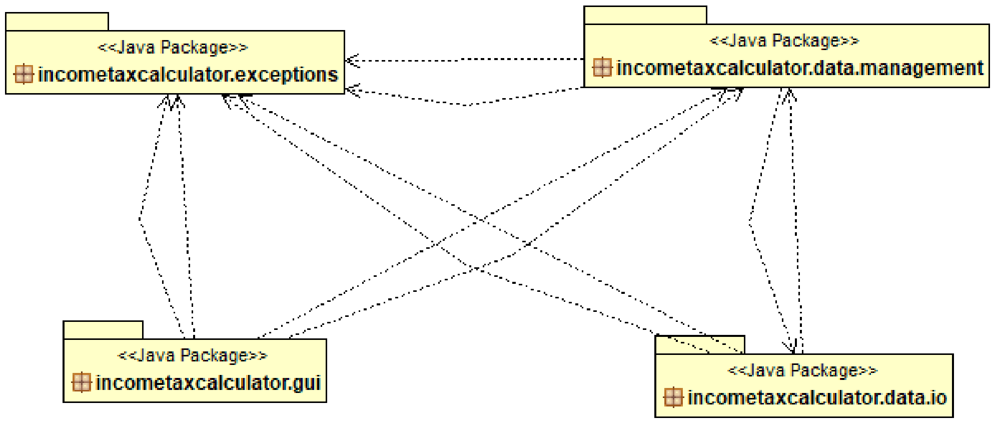
Use case ID	UC9

Actors	User
Pre conditions	The application is running
Main flow of events	1. The use case starts when the user clicks the Create Taxpayer button 2. A dialog appears and asks the user for input 3. The user fills the form correctly and clicks ok 3.1. If the Tax Registration Number already exists, the user is notified 3.1.1. The user is asked if they wish to proceed 3.1.1.1. If the user clicks ok, the existing file is updated with the new information 3.1.1.2. If the user clicks no, the user is returned to the main screen 3.2. If the Tax Registration Number associated file does not exist, a new _INFO.txt file is created and is loaded to the list 3.3. If the Tax Registration Number associated file exists but is not loaded on the list the user is asked if they wish to proceed 3.3.1. If the user clicks ok, the existing file is updated with the new information 3.3.2. If the user clicks no, the user is returned to the main screen
Post conditions	The user returns to the main screen successfully

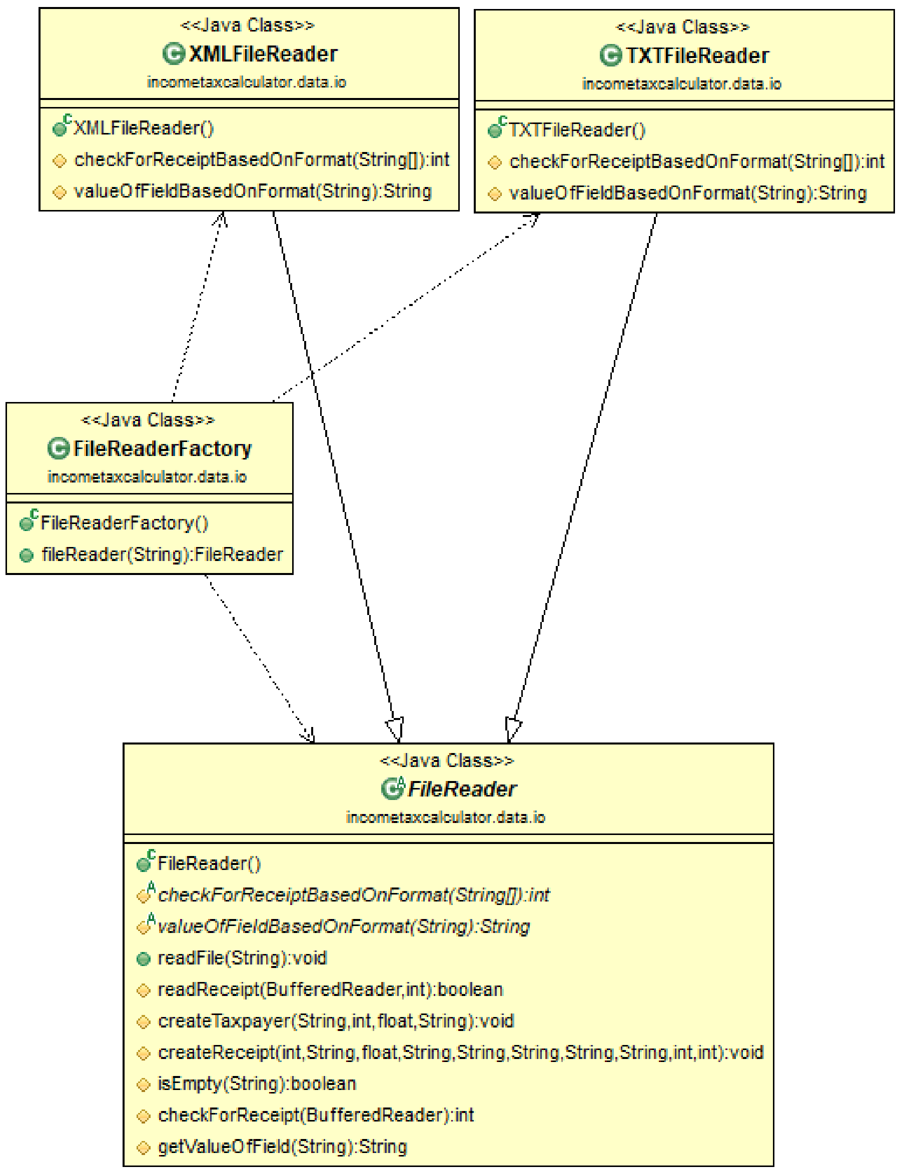
<Use Case 10: Search Taxpayer>

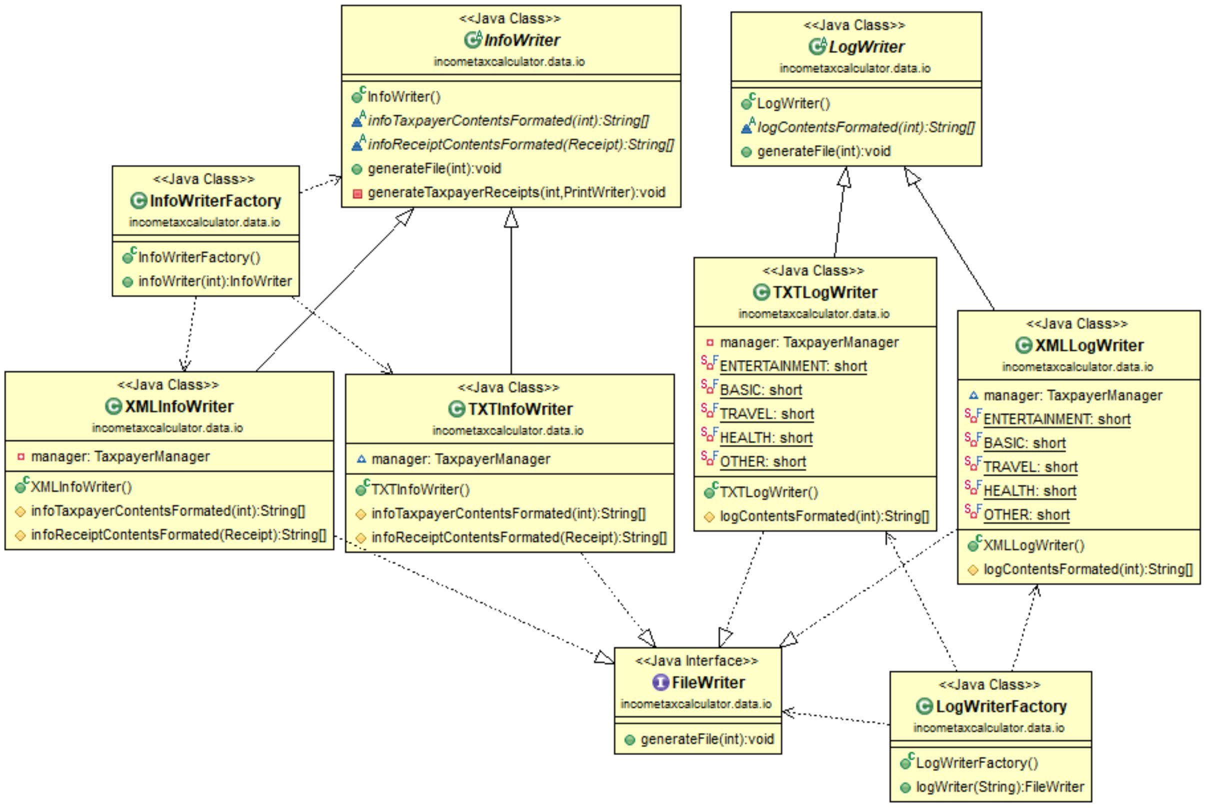
Use case ID	UC10
Actors	User
Pre conditions	The application is running and the list of taxpayers is not empty
Main flow of events	1. The use case starts when the user clicks the search text field 2. The user types to the text field 3. The list displays the already loaded Tax Registration Numbers that contain the search term

ARCHITECTURE

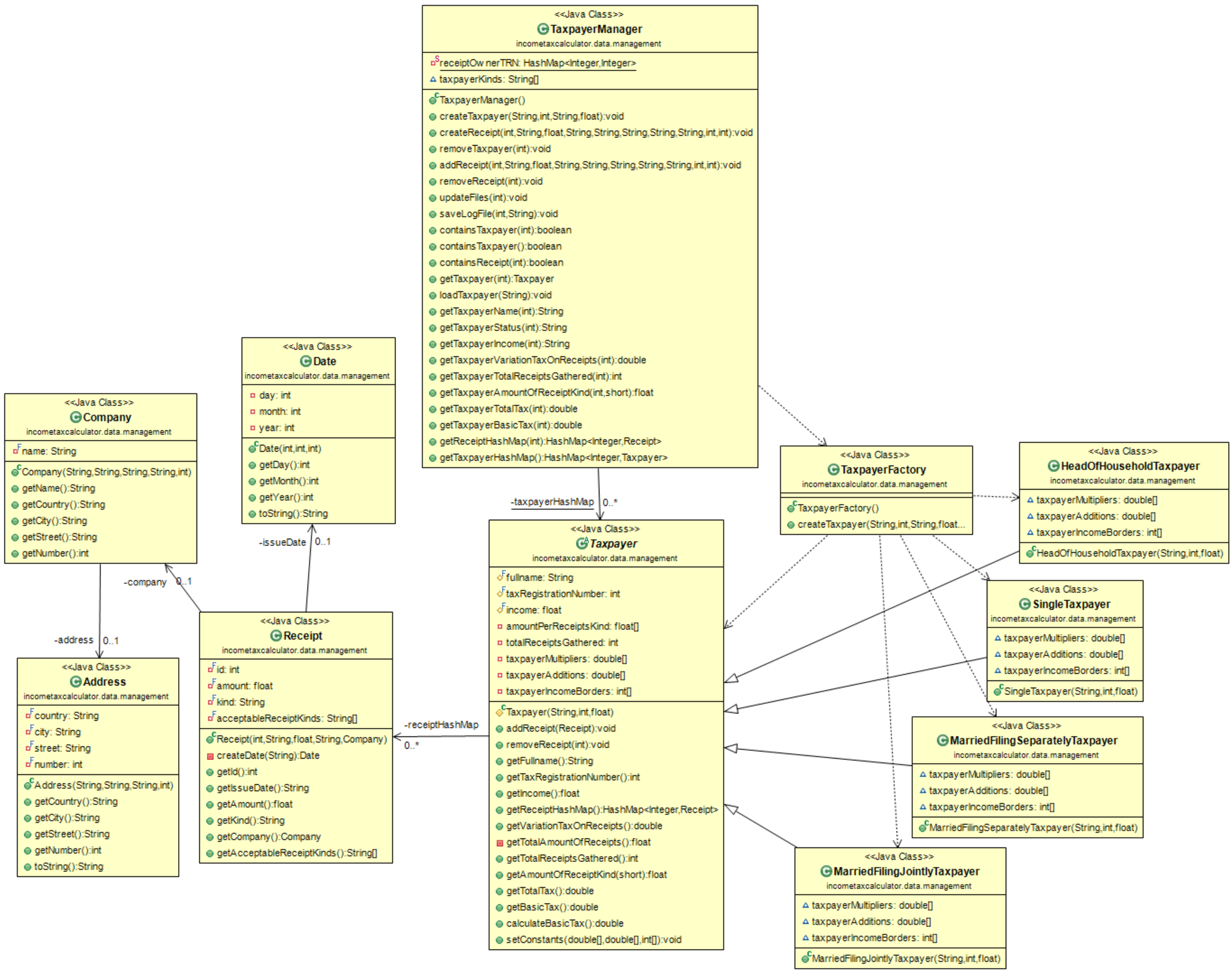

<<Java Package>>
incometaxcalculator.exceptions
<<Java Package>>
incometaxcalculator.data.management
<<Java Package>>
incometaxcalculator.gui
<<Java Package>>
incometaxcalculator.data.io

incometaxcalculator.data.io package

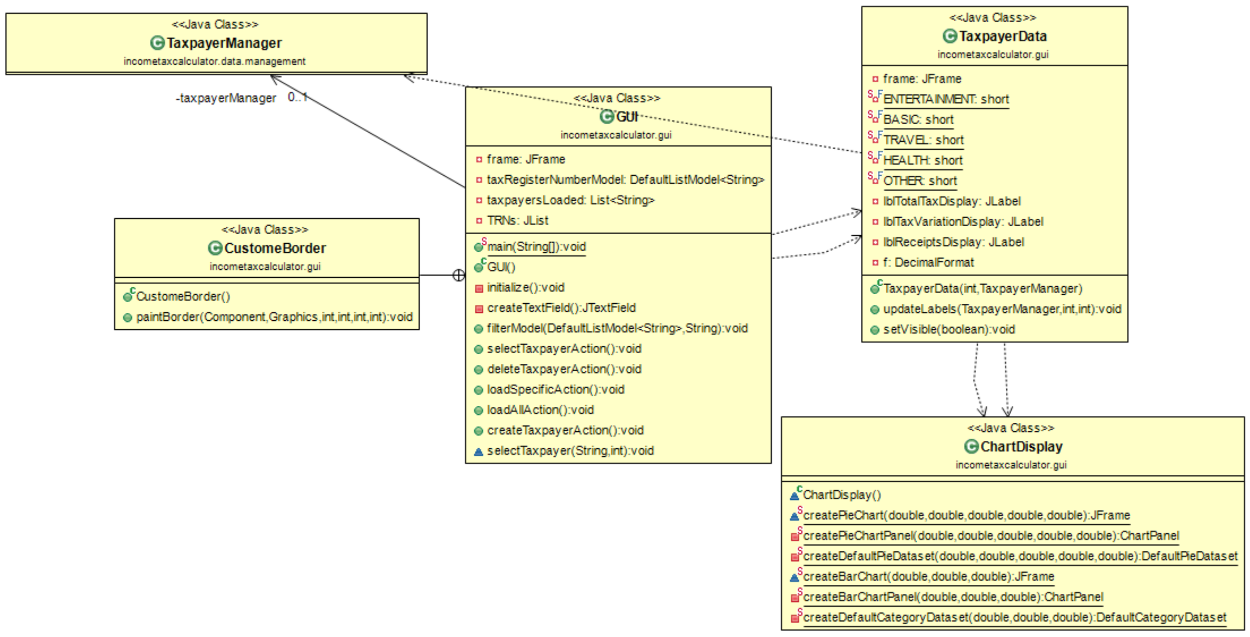

<<Java Class>>
XMLFileReader
incometaxcalculator.data.io
XMLFileReader()
checkForReceiptBasedOnFormat(String[]):int
valueOfFieldBasedOnFormat(String):String
<<Java Class>>
TXTFileReader
incometaxcalculator.data.io
TXTFileReader()
checkForReceiptBasedOnFormat(String[]):int
valueOfFieldBasedOnFormat(String):String
<<Java Class>>
FileReaderFactory
incometaxcalculator.data.io
FileReaderFactory()
fileReader(String):FileReader
<<Java Class>>
FileReader
incometaxcalculator.data.io
FileReader()
checkForReceiptBasedOnFormat(String[]):int
valueOfFieldBasedOnFormat(String):String
readFile(String):void
readReceipt(BufferedReader,int):boolean
createTaxpayer(String,int,float,String):void
createReceipt(int,String,float,String,String,String,String,String,int,int):void
isEmpty(String):boolean
checkForReceipt(BufferedReader):int
getValueOfField(String):String


<<Java Class>>
InfoWriter
incometaxcalculator.data.io
InfoWriter()
infoTaxpayerContentsFormated(int):String[]
infoReceiptContentsFormated(Receipt):String[]
generateFile(int):void
generateTaxpayerReceipts(int,PrintWriter):void
<<Java Class>>
LogWriter
incometaxcalculator.data.io
LogWriter()
logContentsFormated(int):String[]
generateFile(int):void
<<Java Class>>
InfoWriterFactory
incometaxcalculator.data.io
InfoWriterFactory()
infoWriter(int):InfoWriter
<<Java Class>>
TXTLogWriter
incometaxcalculator.data.io
manager: TaxpayerManager
ENTERTAINMENT: short
BASIC: short
TRAVEL: short
HEALTH: short
OTHER: short
TXTLogWriter()
logContentsFormated(int):String[]
<<Java Class>>
XMLLogWriter
incometaxcalculator.data.io
manager: TaxpayerManager
ENTERTAINMENT: short
BASIC: short
TRAVEL: short
HEALTH: short
OTHER: short
XMLLogWriter()
logContentsFormated(int):String[]
<<Java Class>>
XMLInfoWriter
incometaxcalculator.data.io
manager: TaxpayerManager
XMLInfoWriter()
infoTaxpayerContentsFormated(int):String[]
infoReceiptContentsFormated(Receipt):String[]
<<Java Class>>
TXTInfoWriter
incometaxcalculator.data.io
manager: TaxpayerManager
TXTInfoWriter()
infoTaxpayerContentsFormated(int):String[]
infoReceiptContentsFormated(Receipt):String[]
<<Java Interface>>
FileWriter
incometaxcalculator.data.io
generateFile(int):void
<<Java Class>>
LogWriterFactory
incometaxcalculator.data.io
LogWriterFactory()
logWriter(String):FileWriter

incometaxcalculator.data.management package

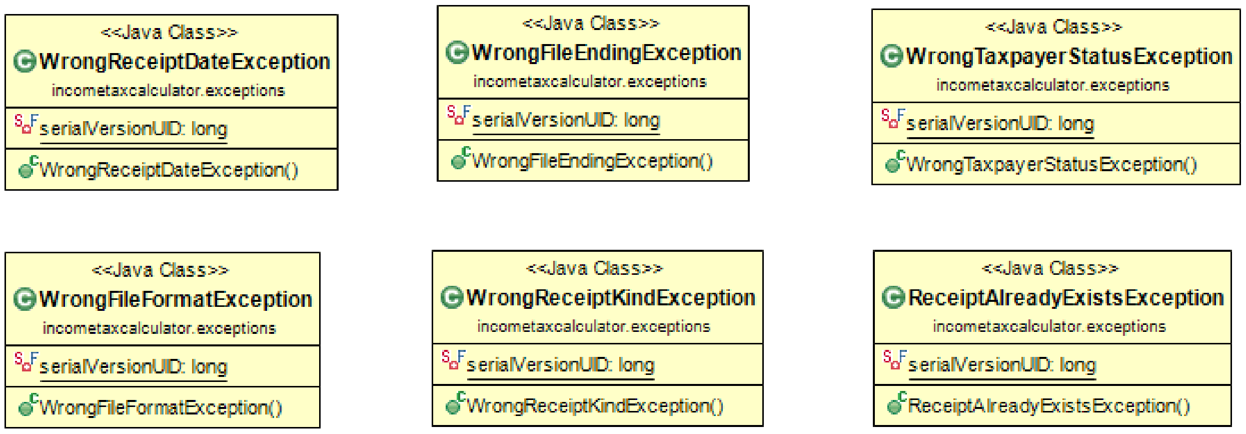

<<Java Class>>
TaxpayerManager
incometaxcalculator.data.management
receiptOwnerTRN: HashMap<Integer,Integer>
taxpayerKinds: String[]
TaxpayerManager()
createTaxpayer(String,int,String,float):void
createReceipt(int,String,float,String,String,String,String,int,int):void
removeTaxpayer(int):void
addReceipt(int,String,float,String,String,String,String,int,int):void
removeReceipt(int):void
updateFiles(int):void
saveLogFile(int,String):void
containsTaxpayer(int):boolean
containsTaxpayer():boolean
containsReceipt(int):boolean
getTaxpayer(int):Taxpayer
loadTaxpayer(String):void
getTaxpayerName(int):String
getTaxpayerStatus(int):String
getTaxpayerIncome(int):String
getTaxpayerVariationTaxOnReceipts(int):double
getTaxpayerTotalReceiptsGathered(int):int
getTaxpayerAmountOfReceiptKind(int,short):float
getTaxpayerTotalTax(int):double
getTaxpayerBasicTax(int):double
getReceiptHashMap(int):HashMap<Integer,Receipt>
getTaxpayerHashMap():HashMap<Integer,Taxpayer>
<<Java Class>>
Date
incometaxcalculator.data.management
day: int
month: int
year: int
Date(int,int,int)
getDay():int
getMonth():int
getYear():int
toString():String
<<Java Class>>
Company
incometaxcalculator.data.management
name: String
Company(String,String,String,String,int)
getName():String
getCountry():String
getCity():String
getStreet():String
getNumber():int
-issueDate
0..1
-company
0..1
-address
0..1
<<Java Class>>
Address
incometaxcalculator.data.management
country: String
city: String
street: String
number: int
Address(String,String,String,int)
getCountry():String
getCity():String
getStreet():String
getNumber():int
toString():String
<<Java Class>>
Receipt
incometaxcalculator.data.management
id: int
amount: float
kind: String
acceptableReceiptKinds: String[]
Receipt(int,String,float,String,Company)
createDate(String):Date
getId():int
getIssueDate():String
getAmount():float
getKind():String
getCompany():Company
getAcceptableReceiptKinds():String[]
-receiptHashMap
0..*
-taxpayerHashMap
0..*
<<Java Class>>
Taxpayer
incometaxcalculator.data.management
fullname: String
taxRegistrationNumber: int
income: float
amountPerReceiptsKind: float[]
totalReceiptsGathered: int
taxpayerMultipliers: double[]
taxpayerAdditions: double[]
taxpayerIncomeBorders: int[]
Taxpayer(String,int,float)
addReceipt(Receipt):void
removeReceipt(int):void
getFullname():String
getTaxRegistrationNumber():int
getIncome():float
getReceiptHashMap():HashMap<Integer,Receipt>
getVariationTaxOnReceipts():double
getTotalAmountOfReceipts():float
getTotalReceiptsGathered():int
getAmountOfReceiptKind(short):float
getTotalTax():double
getBasicTax():double
calculateBasicTax():double
setConstants(double[],double[],int[]):void
<<Java Class>>
TaxpayerFactory
incometaxcalculator.data.management
TaxpayerFactory()
createTaxpayer(String,int,String,float...
<<Java Class>>
HeadOfHouseholdTaxpayer
incometaxcalculator.data.management
taxpayerMultipliers: double[]
taxpayerAdditions: double[]
taxpayerIncomeBorders: int[]
HeadOfHouseholdTaxpayer(String,int,float)
<<Java Class>>
SingleTaxpayer
incometaxcalculator.data.management
taxpayerMultipliers: double[]
taxpayerAdditions: double[]
taxpayerIncomeBorders: int[]
SingleTaxpayer(String,int,float)
<<Java Class>>
MarriedFilingSeparatelyTaxpayer
incometaxcalculator.data.management
taxpayerMultipliers: double[]
taxpayerAdditions: double[]
taxpayerIncomeBorders: int[]
MarriedFilingSeparatelyTaxpayer(String,int,float)
<<Java Class>>
MarriedFilingJointlyTaxpayer
incometaxcalculator.data.management
taxpayerMultipliers: double[]
taxpayerAdditions: double[]
taxpayerIncomeBorders: int[]
MarriedFilingJointlyTaxpayer(String,int,float)

incometaxcalculator.gui



<<Java Class>>
TaxpayerManager
incometaxcalculator.data.management
-taxpayerManager 0..1
<<Java Class>>
GUI
incometaxcalculator.gui
frame: JFrame
taxRegisterNumberModel: DefaultListModel<String>
taxpayersLoaded: List<String>
TRNs: JList
main(String[]):void
GUI()
initialize():void
createTextField():JTextField
filterModel(DefaultListModel<String>,String):void
selectTaxpayerAction():void
deleteTaxpayerAction():void
loadSpecificAction():void
loadAllAction():void
createTaxpayerAction():void
selectTaxpayer(String,int):void
<<Java Class>>
CustomeBorder
incometaxcalculator.gui
CustomeBorder()
paintBorder(Component,Graphics,int,int,int,int):void
<<Java Class>>
TaxpayerData
incometaxcalculator.gui
frame: JFrame
ENTERTAINMENT: short
BASIC: short
TRAVEL: short
HEALTH: short
OTHER: short
lblTotalTaxDisplay: JLabel
lblTaxVariationDisplay: JLabel
lblReceiptsDisplay: JLabel
f: DecimalFormat
TaxpayerData(int,TaxpayerManager)
updateLabels(TaxpayerManager,int,int):void
setVisible(boolean):void
<<Java Class>>
ChartDisplay
incometaxcalculator.gui
ChartDisplay()
createPieChart(double,double,double,double,double):JFrame
createPieChartPanel(double,double,double,double,double):ChartPanel
createDefaultPieDataset(double,double,double,double,double):DefaultPieDataset
createBarChart(double,double,double):JFrame
createBarChartPanel(double,double,double):ChartPanel
createDefaultCategoryDataset(double,double,double):DefaultCategoryDataset

incometaxcalculator.exceptions



<<Java Class>>
WrongReceiptDateException
incometaxcalculator.exceptions
serialVersionUID: long
WrongReceiptDateException()
<<Java Class>>
WrongFileEndingException
incometaxcalculator.exceptions
serialVersionUID: long
WrongFileEndingException()
<<Java Class>>
WrongTaxpayerStatusException
incometaxcalculator.exceptions
serialVersionUID: long
WrongTaxpayerStatusException()
<<Java Class>>
WrongFileFormatException
incometaxcalculator.exceptions
serialVersionUID: long
WrongFileFormatException()
<<Java Class>>
WrongReceiptKindException
incometaxcalculator.exceptions
serialVersionUID: long
WrongReceiptKindException()
<<Java Class>>
ReceiptAlreadyExistsException
incometaxcalculator.exceptions
serialVersionUID: long
ReceiptAlreadyExistsException()

Addressing the different problems of the old design

incometaxcalculator.data.management package

- **Company class :** Removed dead code

- **Taxpayer class** : Simplified complex conditional logic of methods addReceipt(), removeReceipt(), getVariationTaxOnReceipts() by using for loops

- **Subclasses of the Taxpayer class** : Used arrays to store the different constants in the subclasses. Changed calculateBasicTax() to use these arrays. This resulted in having the same method in every subclass. The method was pulled up to the base class, consequently the code duplication was removed

- **TaxpayerManager class :** delegated responsibilities to subordinate classes

    - createTaxpayer() : moved conditional logic into a new class named TaxpayerFactory. The new class is a parameterized factory that creates Taxpayer objects and returns them to the caller. Called the new class from the method

    - updateFiles() : moved conditional logic into a new class named InfoWriterFactory. The new class is a parameterized factory that creates InfoWriter objects and returns them to the caller. Called the new class from the method

    - saveLogFile() : moved common parts out of the complex if-else logic. Then moved the conditional logic to a new class named LogWriteractory. The new class is a parameterized factory that creates LogWriter objects and returns them to the caller. Called the new class from the method

    - loadTaxpayer() : moved common parts out of the complex if-else logic. Then moved the conditional logic to a new class named FileReaderFactory. The new class is a parameterized factory that created FileWriter objects and returns them to the caller. Called the new class from the method

incometaxcalculator.data.io package

- **TXTFileReader, XMLFileReader classes :** Located methods that were similar. Extracted the parts of the code that were **different** and moved them in new simple methods named

  checkForReceiptBasedOnFormat(String values[])

  valueOfFieldBasedOnFormat(String fieldsLine)

  The extraction made the similar methods identical. Then the identical methods were pulled up to the base FileReader class.

  Lastly, defined abstract methods in the base class that correspond to the two simple methods that were created

- **FileWriter class :**

    - removed methods that simply delegated calls to respective methods in TaxpayerManager class, hence removed the Middle Man

    - removed methods that weren't being used by all of the subclasses by pushing them down to the subclasses that needed them, hence removed the problem of Refuse Bequest

    - The FileWriter class was converted to an interface

- **TXTInfoWriter, XMLInfoWriter classes :**
    - Created a new abstract class InfoWriter that implements the FileWriter interface
    - Extend InfoWriter from TXTInfoWriter and XMLInfoWriter
    - Created two template methods in InfoWriter, generateFile() and generateTaxpayerReceipts()
    - Created two abstract methods in InfoWriter, the methods' return type is a string array
    - The two subclasses implement the two abstract methods. The methods create a formatted string array to be used by InfoWriter's template methods.
- **TXTLogWriter, XMLLogWriter classes :**
    - Created a new abstract class LogWriter that implements the FileWriter interface
    - Extend LogWriter from TXTLogWriter and XMLLogWriter
    - Created one template method in LogWriter, generateFile()
    - Created one abstract method in LogWriter, logContentsFormatted(). The method's return type is a string array
    - The two subclasses implement the abstract method. The method in each one create a formatted string that will be used by the LogWriter's template method

CLASSES RESPONSIBILITIES AND COLLABORATIONS (CRC CARDS)

- For each class give a brief description in terms of a CRC card (see the format below)

PACKAGE incometaxcalculator.data.io

Class Name: FileReader	
Responsibilities	**Collaborations**
• Reads file and inputs the contents to a TaxpayerManager object accordingly	• Creates a TaxpayerManager object • XMLFileReader and TXTFileReader extend FileReader

Interface Name: FileWriter	
Responsibilities	**Collaborations**
• Declares abstract method that generates files	• XMLInfoWriter, TXTInfoWriter, TXTLogWriter, XMLLogWriter, LogWriter, InfoWriter implement FileWriter

Interface Name: InfoWriter	
Responsibilities	**Collaborations**

• Contains a template methods that store a taxpayer's information in a file	• Implements the **FileWriter** interface • Has two subclasses, **XMLInfoWriter** and **TXTInfoWriter** • Creates **TaxpayerManager** object to gain access to the contents of a hashmap. • Creates a hashmap<Integer, **Receipt**> to gain access to the receipts of the taxpayer

Class Name: InfoWriterFactory	
Responsibilities	**Collaborations**
• Parameterized factory that creates InfoWriter objects	• Creates a **TXTInfoWriter** or an **XMLInfoWriter** object

Class Name: FileReaderFactory	
Responsibilities	**Collaborations**
• Parameterized factory that creates FileReader objects	• Creates a **TXTFileReader** or an **XMLFileReader** object

Class Name: LogWriter	
Responsibilities	**Collaborations**
• Contains a template method that generates a log file that stores the taxpayer's tax information	• Has two subclasses, **TXTLogWriter** and **XMLLogWriters** • Creates a TaxpayerManager object to gain access to a taxpayer's information

Class Name: LogWriterFactory	
Responsibilities	**Collaborations**
• Parameterized factory that creates LogReader objects	• Creates a **TXTLogWriter** or an **XMLLogWriter** object

Class Name: TXTFileReader	
Responsibilities	**Collaborations**
• Returns receipt information to the caller based on the txt file's layout	• extends **FileReader**

Class Name: XMLFileReader	
Responsibilities	**Collaborations**
• Returns receipt information to the caller based on the xml file's layout	• extends **FileReader**

Class Name: TXTInfoWriter	
Responsibilities	**Collaborations**
• Creates arrays according to the _INFO.txt file layout. • Returns these arrays to the extended parent class.	• Extends **InfoWriter** • Implements **FileWriter** • Create a **TaxpayerManager** object to gain access to a taxpayer's information

Class Name: XMLInfoWriter	
Responsibilities	**Collaborations**
• Creates arrays according to the _INFO.xml file layout. • Returns these arrays to the extended parent class.	• Extends **InfoWriter** • Implements **FileWriter** • Create a **TaxpayerManager** object to gain access to a taxpayer's information

Class Name: TXTLogWriter	
Responsibilities	**Collaborations**
• Creates arrays according to the _LOG.txt file layout. • Returns these arrays to the extended parent class.	• Extends **LogWriter** • Implements **FileWriter** • Create a **TaxpayerManager** object to gain access to a taxpayer's information

Class Name: XMLLogWriter	
Responsibilities	**Collaborations**
• Creates arrays according to the _LOG.xml file layout. • Returns these arrays to the extended parent class.	• Extends **LogWriter** • Implements **FileWriter** • Create a **TaxpayerManager** object to gain access to a taxpayer's information

PACKAGE incometaxcalculator.data.management

Class Name: Address	
Responsibilities	**Collaborations**

• Contains getters that return the object's information	•

Class Name: Company	
Responsibilities	**Collaborations**
• Contains getters that return the objects's information	•

Class Name: Date	
Responsibilities	**Collaborations**
• Contains getters that return the objects's information	•

Class Name: Receipt	
Responsibilities	**Collaborations**
• Checks a date given as a parameter, if it has the correct format, creates a Date object • Contains getters that return the object's	• Constructor takes a **Company** object parameter • Creates a **Date** object

information	

Class Name: Taxpayer	
Responsibilities	**Collaborations**
• Handles a taxpayer’s information • Adds receipts to the taxpayer • Removes receipts from the taxpayer • Calculates the basic tax • Calculates the receipts’ tax • Calculates the total tax	•

Class Name: HeadOfHouseholdTaxpayer	
Responsibilities	**Collaborations**
• Sets the extended class’s name, tax registration number and income • Sets the extended class’s constants. The super class uses these constants to	• Extends **Taxpayer**

calculate taxes	

Class Name: MarriedFilingJointlyTaxpayer	
Responsibilities	**Collaborations**
• Sets the extended class's name, tax registration number and income • Sets the extended class's constants. The super class uses these constants to calculate taxes	• Extends **Taxpayer**

Class Name: MarriedFilingSeparatelyTaxpayer	
Responsibilities	**Collaborations**
• Sets the extended class's name, tax registration number and income • Sets the extended class's constants. The super class uses these constants to calculate taxes	• Extends **Taxpayer**

Class Name: SingleTaxpayer	
Responsibilities	**Collaborations**
• Sets the extended class's name, tax registration number and income • Sets the extended class's constants. The super class uses these constants to calculate taxes	• Extends **Taxpayer**

Class Name: TaxpayerFactory	
Responsibilities	**Collaborations**
• Parameterized factory that creates **Taxpayer** objects	• Creates **HeadOfHouseholdTaxpayer, MarriedFilingJointlyTaxpayer, MarriedFilingSeparatelyTaxpayer, SingleTaxpayer** objects

Class Name: TaxpayerManager	
Responsibilities	**Collaborations**
• Manages taxpayers • Has a hashmap that stores the multiple	• Calls a **TaxpayerFactory** to get a **Taxpayer** • Creates **Receipt** object

Taxpayer objects • Has a hashap that stores receipts • Creates a **Taxpayer** object by calling TaxpayerFactory. Adds object to hashmap • Creates a **Receipt** object. Adds object to receipt hashmap and adds the receipt to the corresponding taxpayer • Removes taxpayer from the taxpayer hashmap • Adds receipts after checking for duplicates and updates corresponding files • Removes receipt from hashmap and updates corresponding files • Saves _LOG files • Loads taxpayers from files by calling FileReaderFactory	• Calles **InfoWriterFactory** to get an **InfoWriter** • Calls a **LogWriterFactory** to get a **LogWriter** • Calles **FileReaderFactory** to get a **FileReader**

PACKAGE incometaxcalculator.gui

Class Name: GUI	
Responsibilities	**Collaborations**

• Creates the main screen of the application • Calls TaxpayerData if the user wishes to view a specific taxpayer	• Creates a **TaxpayerManager** object • Creates a **TaxpayerData** object

Class Name: TaxpayerData	
Responsibilities	**Collaborations**
• Creates the selected taxpayer's screen of the application • Displays selected taxpayer's information • Provides user with a button to view the taxpayer's charts • Can add and delete receipts • Can save the data	• The constructor needs a **TaxpayerManager** parameter • Has a **Receipt** hashmap to handle the taxpayer's receipts • Calls **ChartDisplay**

Class Name: ChartDisplay	
Responsibilities	**Collaborations**
• Displays bar chart	•

• Displays pie chart	