INCOME TAX CALCULATOR

OVERALL REPORT

VERSION <1.0>

Κωνσταντίνος Ανδρέου 4316

TABLE OF CONTENTS

Introduction 4

Refactored Design 4

- Use Cases 4
- Architecture 8
- Detailed Design 8
- Classes Responcibilities and Collaborations (CRC CARDS) 14

INTRODUCTION

The objective of the report is to provide a refactored design of the Minnesota Income Tax Calculator application.

REFACTORED DESIGN

USE CASES

Specify the use cases of the application.

Use case ID	UC1: LoadTaxpayerInfo
Actors	User
Pre conditions	-
Main flow of events	1. The use case starts when the user selects Load Taxpayer option from the menu. 2. The system asks for the tax registration number and the file's type. 3. The user selects between the file format options and enters the registration number.
Alternative flow	1. User enters an invalid registration number. 2. The system asks again for the tax registration number.
Post conditions	The taxpayer is loaded in the system.

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 selects Select Taxpayer option from the menu. 2. The system asks for the registration number of a taxpayer. 3. The user types a taxpayer's registration number and clicks OK. 4. The system opens a new window where user can access information of the requested taxpayer.
Alternative flow	1. The user enters a registration number that isn't loaded in system. 2. The system shows an error message and returns to the previous state.
Post conditions	The taxpayer is selected and a menu with new options pops.

Use case ID	UC3: AddReceipt
Actors	User
Pre conditions	A taxpayer must be selected.
Main flow of events	1. The use case starts when the user selects Add Receipt option from the menu. 2. The system opens a form where user can give receipt information(Receipt ID, Date, Kind, etc). 3. The user fills the form with the information about a receipt. 4. The system adds the receipt in taxpayer's INFO file.
Alternative flow	1. User enters wrong information at a receipt' s field or receipt already exists. 2. The system handles the error by not adding the receipt.

Post conditions	The system refreshes the taxpayer’s file with the new receipt.

Use case ID	UC4: DeleteReceipt
Actors	User
Pre conditions	A taxpayer must be selected.
Main flow of events	1. The use case starts when the user selects Delete Receipt option from the menu. 2. The system asks for the receipt’s ID. 3. The user fills the receipt’s ID. 4. The system deletes the receipt from the taxpayer’s INFO file.
Alternative flow	1. The user gives a wrong receipt ID. 2. The system handles the error and does not change the state of taxpayer’s INFO file.
Post conditions	The system refreshes the taxpayer’s file by removing the receipt.

Use case ID	UC5: ViewReport
Actors	User
Pre conditions	A taxpayer must be selected.
Main flow of events	1. The use case starts when the user selects View Report from the menu. 2. The system shows a bar chart of the basic tax, the increase or decrease due to receipts and the total tax and a pie chart of the percentage of the total amount of each kind of receipt.

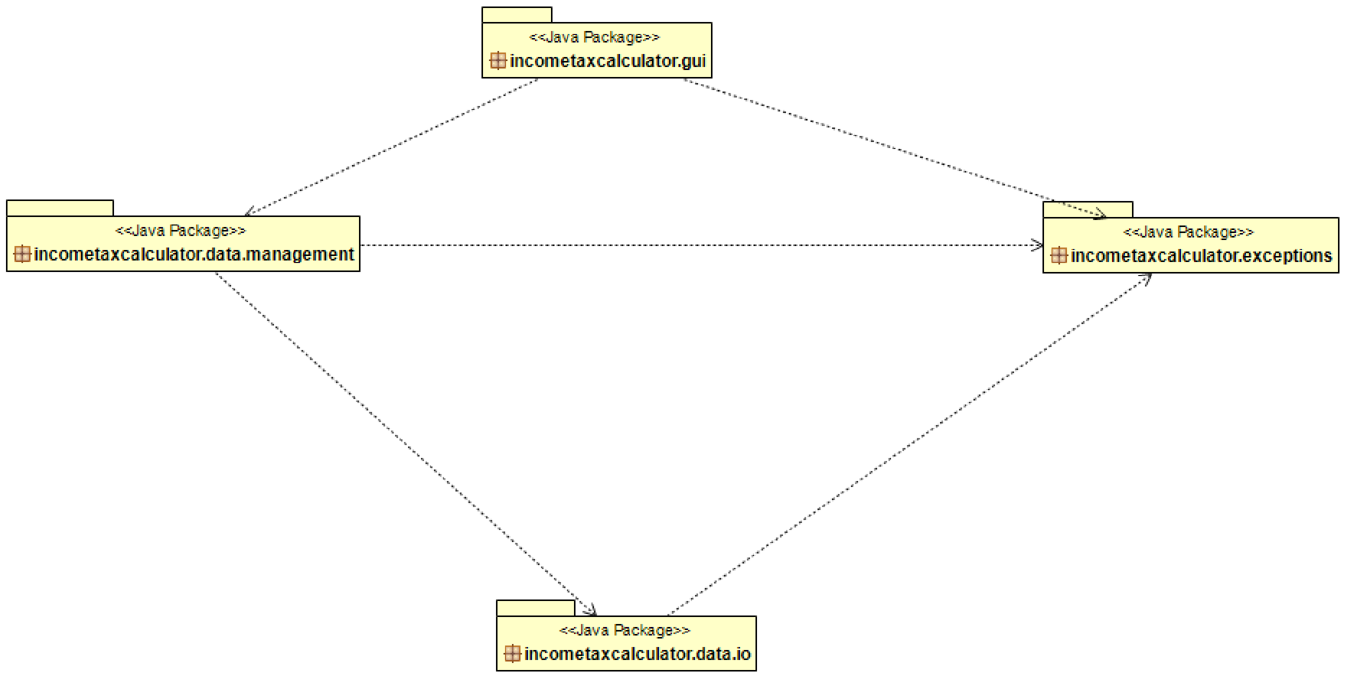
Alternative flow	-
Post conditions	-

Use case ID	UC6: SaveToFile
Actors	User
Pre conditions	-
Main flow of events	1. The use case starts when the user selects Save Data option from the menu 2. The system opens a window where user can choose between text and xml file format. 3. The user selects a file format. 4. The system saves the taxpayer's data in a file.
Alternative flow	-
Post conditions	The taxpayer's report is saved into the system.

Use case ID	UC7: DeleteTaxpayer
Actors	User
Pre conditions	A taxpayer must be loaded into the system.

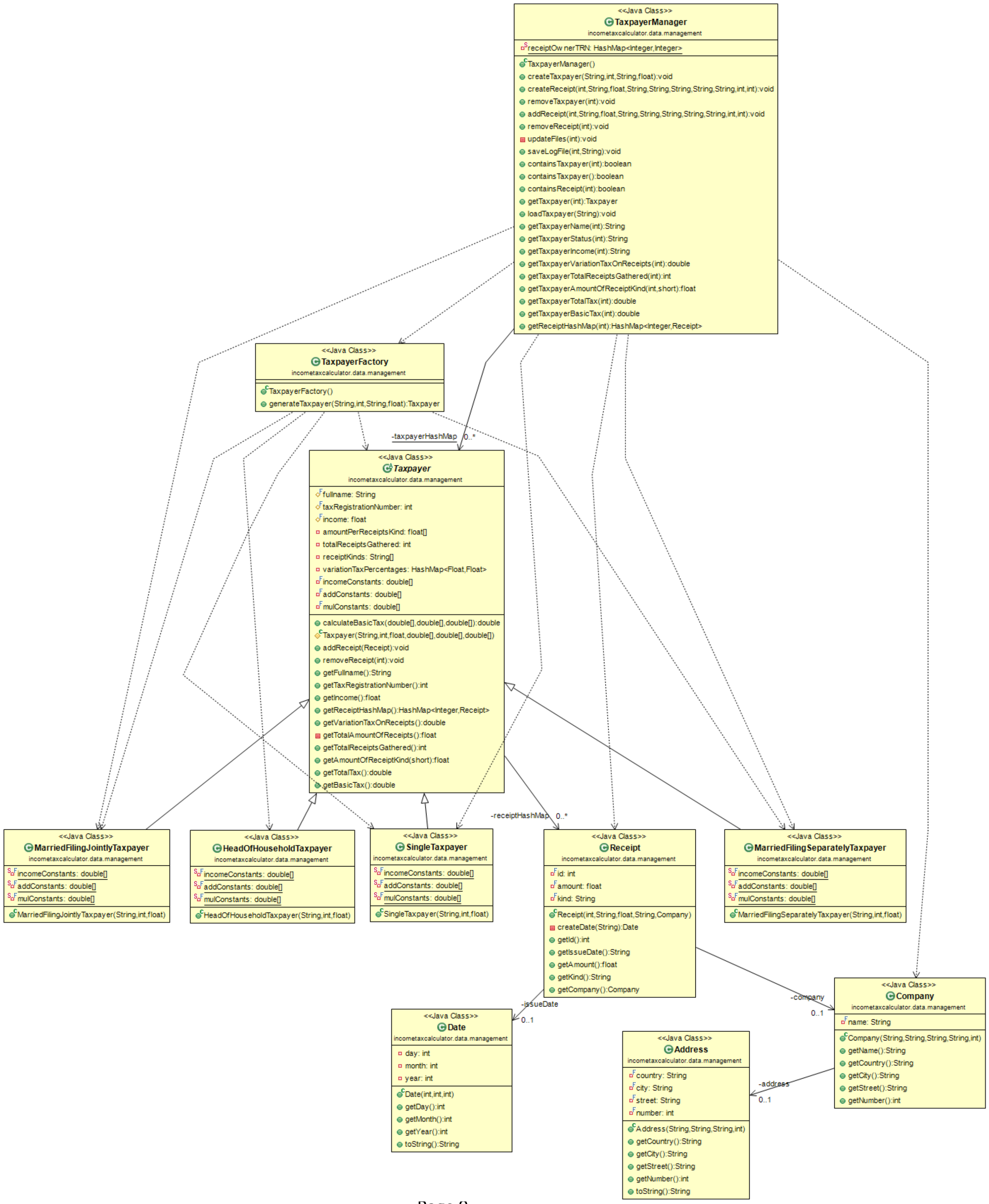
Main flow of events	1. The use case starts when the user selects Delete Taxpayer option from the menu. 2. The system asks for the registration number of a taxpayer. 3. The user types a taxpayer's registration number and clicks OK. 4. The system deletes the taxpayer from the list of taxpayers.
Alternative flow	1. The user enters a registration number that isn't loaded in system. 2. The system handles the error and does not delete any taxpayer from the list.
Post conditions	The system has removed the taxpayer.

ARCHITECTURE


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

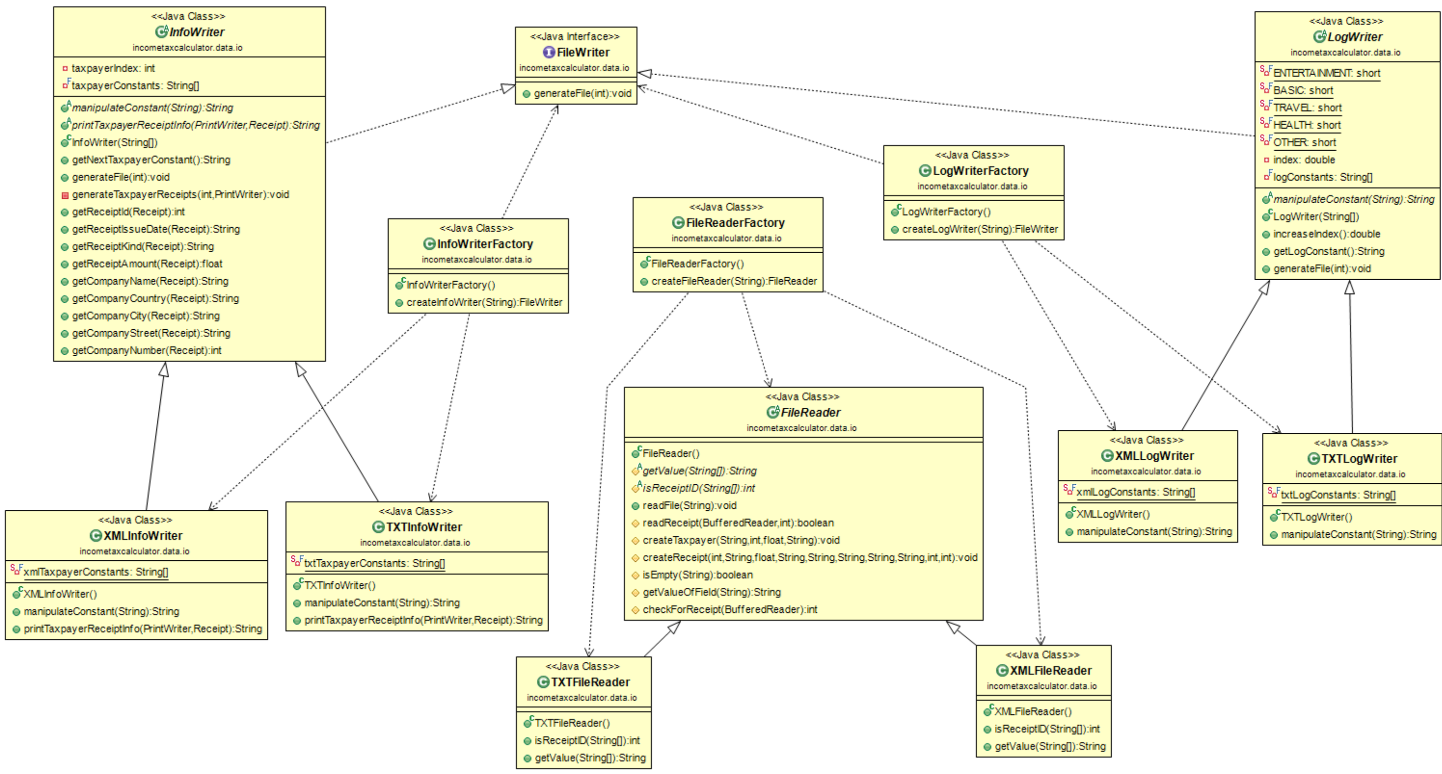
DETAILED DESIGN

incometaxcalculator.data.management package



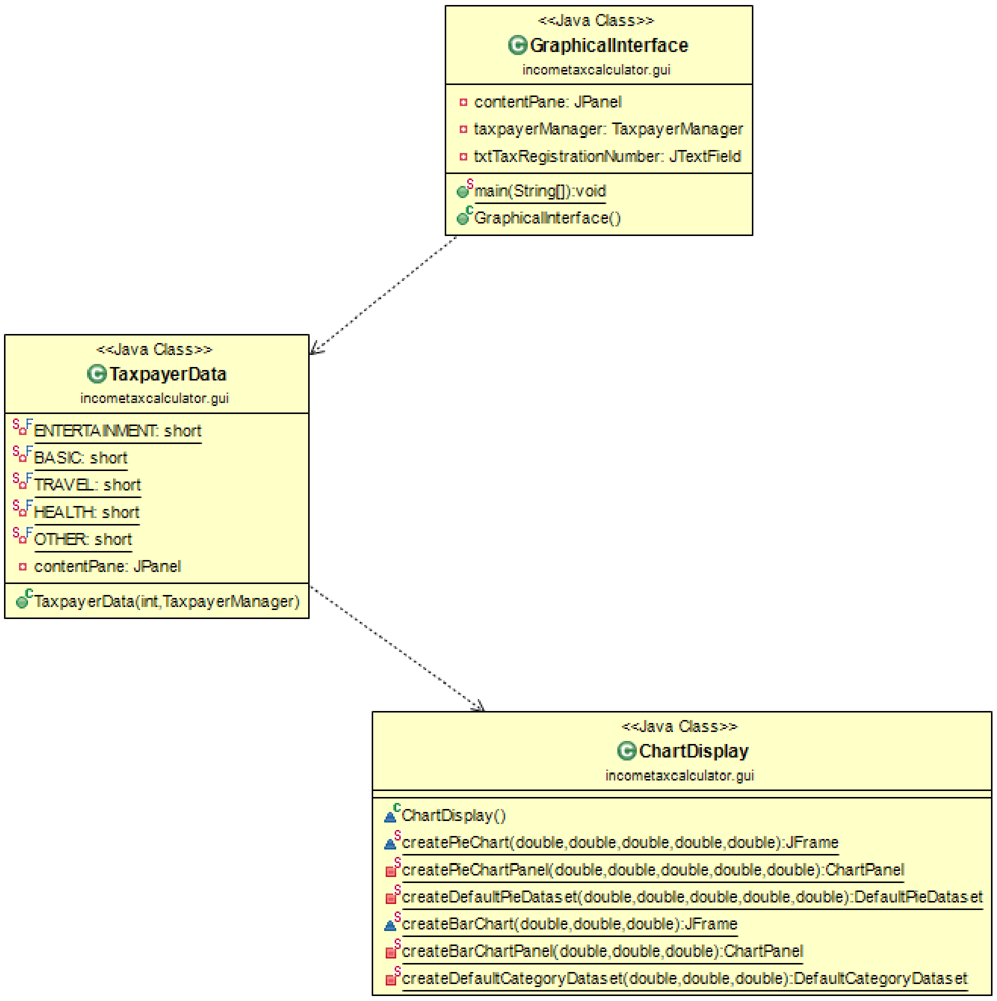
<<Java Class>>
TaxpayerManager
incometaxcalculator.data.management
receiptOwnerTRN: HashMap<Integer,Integer>
TaxpayerManager()
createTaxpayer(String,int,String,float):void
createReceipt(int,String,float,String,String,String,String,String,int,int):void
removeTaxpayer(int):void
addReceipt(int,String,float,String,String,String,String,String,int,int):void
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>
<<Java Class>>
TaxpayerFactory
incometaxcalculator.data.management
TaxpayerFactory()
generateTaxpayer(String,int,String,float):Taxpayer
-taxpayerHashMap
0..*
<<Java Class>>
Taxpayer
incometaxcalculator.data.management
fullname: String
taxRegistrationNumber: int
income: float
amountPerReceiptsKind: float[]
totalReceiptsGathered: int
receiptKinds: String[]
variationTaxPercentages: HashMap<Float,Float>
incomeConstants: double[]
addConstants: double[]
mulConstants: double[]
calculateBasicTax(double[],double[],double[]):double
Taxpayer(String,int,float,double[],double[],double[])
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
-receiptHashMap
0..*
<<Java Class>>
MarriedFilingJointlyTaxpayer
incometaxcalculator.data.management
incomeConstants: double[]
addConstants: double[]
mulConstants: double[]
MarriedFilingJointlyTaxpayer(String,int,float)
<<Java Class>>
HeadOfHouseholdTaxpayer
incometaxcalculator.data.management
incomeConstants: double[]
addConstants: double[]
mulConstants: double[]
HeadOfHouseholdTaxpayer(String,int,float)
<<Java Class>>
SingleTaxpayer
incometaxcalculator.data.management
incomeConstants: double[]
addConstants: double[]
mulConstants: double[]
SingleTaxpayer(String,int,float)
<<Java Class>>
Receipt
incometaxcalculator.data.management
id: int
amount: float
kind: String
Receipt(int,String,float,String,Company)
createDate(String):Date
getId():int
getIssueDate():String
getAmount():float
getKind():String
getCompany():Company
<<Java Class>>
MarriedFilingSeparatelyTaxpayer
incometaxcalculator.data.management
incomeConstants: double[]
addConstants: double[]
mulConstants: double[]
MarriedFilingSeparatelyTaxpayer(String,int,float)
-issueDate
0..1
<<Java Class>>
Date
incometaxcalculator.data.management
day: int
month: int
year: int
Date(int,int,int)
getDay():int
getMonth():int
getYear():int
toString():String
-company
0..1
<<Java Class>>
Company
incometaxcalculator.data.management
name: String
Company(String,String,String,String,int)
getName():String
getCountry():String
getCity():String
getStreet():String
getNumber():int
-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

incometaxcalculator.data.io package



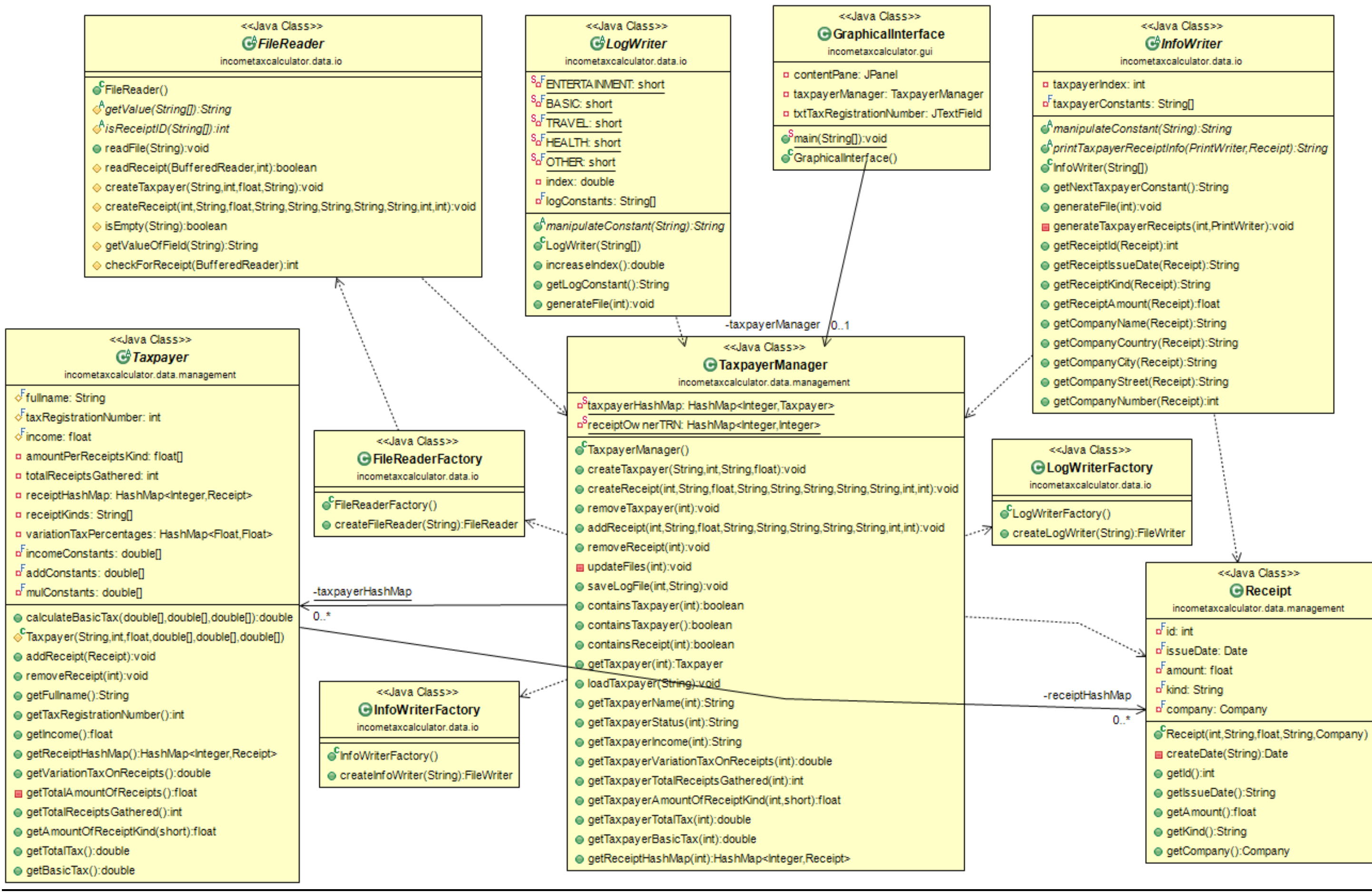
<<Java Class>>
InfoWriter
incometaxcalculator.data.io
taxpayerIndex: int
taxpayerConstants: String[]
manipulateConstant(String):String
printTaxpayerReceiptInfo(PrintWriter,Receipt):String
InfoWriter(String[])
getNextTaxpayerConstant():String
generateFile(int):void
generateTaxpayerReceipts(int,PrintWriter):void
getReceiptId(Receipt):int
getReceiptIssueDate(Receipt):String
getReceiptKind(Receipt):String
getReceiptAmount(Receipt):float
getCompanyName(Receipt):String
getCompanyCountry(Receipt):String
getCompanyCity(Receipt):String
getCompanyStreet(Receipt):String
getCompanyNumber(Receipt):int
<<Java Interface>>
FileWriter
incometaxcalculator.data.io
generateFile(int):void
<<Java Class>>
LogWriter
incometaxcalculator.data.io
ENTERTAINMENT: short
BASIC: short
TRAVEL: short
HEALTH: short
OTHER: short
index: double
logConstants: String[]
manipulateConstant(String):String
LogWriter(String[])
increaseIndex():double
getLogConstant():String
generateFile(int):void
<<Java Class>>
LogWriterFactory
incometaxcalculator.data.io
LogWriterFactory()
createLogWriter(String):FileWriter
<<Java Class>>
InfoWriterFactory
incometaxcalculator.data.io
InfoWriterFactory()
createInfoWriter(String):FileWriter
<<Java Class>>
FileReaderFactory
incometaxcalculator.data.io
FileReaderFactory()
createFileReader(String):FileReader
<<Java Class>>
FileReader
incometaxcalculator.data.io
FileReader()
getValue(String[]):String
isReceiptID(String[]):int
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
getValueOfField(String):String
checkForReceipt(BufferedReader):int
<<Java Class>>
XMLLogWriter
incometaxcalculator.data.io
xmlLogConstants: String[]
XMLLogWriter()
manipulateConstant(String):String
<<Java Class>>
TXTLogWriter
incometaxcalculator.data.io
txtLogConstants: String[]
TXTLogWriter()
manipulateConstant(String):String
<<Java Class>>
XMLInfoWriter
incometaxcalculator.data.io
xmlTaxpayerConstants: String[]
XMLInfoWriter()
manipulateConstant(String):String
printTaxpayerReceiptInfo(PrintWriter,Receipt):String
<<Java Class>>
TXTInfoWriter
incometaxcalculator.data.io
txtTaxpayerConstants: String[]
TXTInfoWriter()
manipulateConstant(String):String
printTaxpayerReceiptInfo(PrintWriter,Receipt):String
<<Java Class>>
TXTFileReader
incometaxcalculator.data.io
TXTFileReader()
isReceiptID(String[]):int
getValue(String[]):String
<<Java Class>>
XMLFileReader
incometaxcalculator.data.io
XMLFileReader()
isReceiptID(String[]):int
getValue(String[]):String

incometaxcalculator.data.gui package



<<Java Class>>
GraphicalInterface
incometaxcalculator.gui
contentPane: JPanel
taxpayerManager: TaxpayerManager
txtTaxRegistrationNumber: JTextField
main(String[]):void
GraphicalInterface()
<<Java Class>>
TaxpayerData
incometaxcalculator.gui
ENTERTAINMENT: short
BASIC: short
TRAVEL: short
HEALTH: short
OTHER: short
contentPane: JPanel
TaxpayerData(int,TaxpayerManager)
<<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

A Class Diagram with the main(top level) classes of the system



<<Java Class>>
FileReader
incometaxcalculator.data.io
FileReader()
getValue(String[]):String
isReceiptID(String[]):int
readFile(String):void
readReceipt(BufferedReader,int):boolean
createTaxpayer(String,int,float,String):void
createReceipt(int,String,float,String,String,String,String,int,int):void
isEmpty(String):boolean
getValueOfField(String):String
checkForReceipt(BufferedReader):int
<<Java Class>>
LogWriter
incometaxcalculator.data.io
ENTERTAINMENT: short
BASIC: short
TRAVEL: short
HEALTH: short
OTHER: short
index: double
logConstants: String[]
manipulateConstant(String):String
LogWriter(String[])
increaseIndex():double
getLogConstant():String
generateFile(int):void
<<Java Class>>
GraphicalInterface
incometaxcalculator.gui
contentPane: JPanel
taxpayerManager: TaxpayerManager
txtTaxRegistrationNumber: JTextField
main(String[]):void
GraphicalInterface()
<<Java Class>>
InfoWriter
incometaxcalculator.data.io
taxpayerIndex: int
taxpayerConstants: String[]
manipulateConstant(String):String
printTaxpayerReceiptInfo(PrintWriter,Receipt):String
InfoWriter(String[])
getNextTaxpayerConstant():String
generateFile(int):void
generateTaxpayerReceipts(int,PrintWriter):void
getReceiptId(Receipt):int
getReceiptIssueDate(Receipt):String
getReceiptKind(Receipt):String
getReceiptAmount(Receipt):float
getCompanyName(Receipt):String
getCompanyCountry(Receipt):String
getCompanyCity(Receipt):String
getCompanyStreet(Receipt):String
getCompanyNumber(Receipt):int
-taxpayerManager
0..1
<<Java Class>>
Taxpayer
incometaxcalculator.data.management
fullname: String
taxRegistrationNumber: int
income: float
amountPerReceiptsKind: float[]
totalReceiptsGathered: int
receiptHashMap: HashMap<Integer,Receipt>
receiptKinds: String[]
variationTaxPercentages: HashMap<Float,Float>
incomeConstants: double[]
addConstants: double[]
mulConstants: double[]
calculateBasicTax(double[],double[],double[]):double
Taxpayer(String,int,float,double[],double[],double[])
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
<<Java Class>>
FileReaderFactory
incometaxcalculator.data.io
FileReaderFactory()
createFileReader(String):FileReader
<<Java Class>>
TaxpayerManager
incometaxcalculator.data.management
taxpayerHashMap: HashMap<Integer,Taxpayer>
receiptOwnerTRN: HashMap<Integer,Integer>
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>
<<Java Class>>
LogWriterFactory
incometaxcalculator.data.io
LogWriterFactory()
createLogWriter(String):FileWriter
-taxpayerHashMap
0..*
<<Java Class>>
InfoWriterFactory
incometaxcalculator.data.io
InfoWriterFactory()
createInfoWriter(String):FileWriter
<<Java Class>>
Receipt
incometaxcalculator.data.management
id: int
issueDate: Date
amount: float
kind: String
company: Company
Receipt(int,String,float,String,Company)
createDate(String):Date
getId():int
getIssueDate():String
getAmount():float
getKind():String
getCompany():Company
-receiptHashMap
0..*

Addressing the different problems of the old design

In the **Taxpayer** class addReceipt(), removeReceipt(), getVariationTaxOnReceipts() methods had chained if-else statements that were checking conditions based on constants. The constants are now stored in simple arrays and the chained if-else statements are replaced with for loops. Another problem in the Taxpayer class was the abstract method calculateBasicTax() that was similar in each subclass. This problem was solved by implementing the method in the Taxpayer class and parameterizing it with arrays storing the different constants for each type of Taxpayer(subclasses) and these arrays became fields of the class that are initialised in the Taxpayer constructor. The subclasses(HeadOfHouseholdTaxpayer, SingleTaxpayer etc.) are now used only to initialize the arrays with constants.

In the **TaxpayerManager** class the conditional logic of creating a Taxpayer object inside createTaxpayer() was moved in a method inside a new TaxpayerFactory class. The updateFiles() method was refactored in a more readable form and the creation of the different InfoWriter objects was moved in InfoWriterFactory class. Similarly, to the previous changes, the conditional logic of creating different kind of objects in the saveLogFile() and loadTaxpayer() was moved in LogWriterFactory and FileReaderFactory respectively.

In the **TXTFileReader, XMLFileReader** classes the two methods checkForReceipt and getValueOfField were similar. This problem was fixed by extracting the different parts of code in simple methods and making these simple methods abstract in the base class moving the implementation of the two methods(checkForReceipt and getValueOfField) in abstract FileReader class.

In the **TXTInfoWriter, XMLInfoWriter** classes, the methods had duplicate code. The two methods were different only in constant string tags. For the private method generateTaxpayerReceipts() the problem was fixed by abstracting the different part in a simple method called printTaxpayerReceiptsInfo() and forming a template method in an abstract InfoWriter super class. For the generateFile() method the different constant strings were put in an array in both subclasses TXTInfoWriter and XMLInfoWriter and a method manipulateConstant() was declared to make them identical. The method generateFile() was moved in InfoWriter superclass where the arrays were initialized by its constructor and the manipulateConstant() method was declared abstract.

The similar changes were made in **TXTLogWriter, XMLLogWriter** where different string constants of generateFile() were initialized in arrays and the method was moved to a new abstract class LogWriter. After all, FileWriter became an interface where generateFile() method is declared that LogWriter and InfoWriter classes implement.

Changes in the **GUI** (GraphicalInterface class) were also made in order to make it user friendly. The load of taxpayers can now be made by using a File Chooser where user selects a txt or xml file by browsing in his/her files. The select and delete taxpayer options were requiring typing his/her AFM. Now, this thing can be done by just clicking on the list of available taxpayers and then click at the select or delete taxpayer option from the menu.

CLASSES RESPONSIBILITIES AND COLLABORATIONS (CRC CARDS)

Class Name: FileReader	
Responsibilities	**Collaborations**
This class is responsibly for reading the taxpayer’s info(_INFO files).	-

Class Name: TXTFileReader	
Responsibilities	**Collaborations**
This class provides simple methods to FileReader for the reading of a txt file.	FileReader

Class Name: XMLFileReader	
Responsibilities	**Collaborations**
This class provides simple methods to FileReader for the reading of a xml file.	FileReader

Class Name: FileReaderFactory	
Responsibilities	**Collaborations**
This class creates a specific FileReader object.	FileReader XMLFileReader TXTFileReader

Class Name: FileWriter	
Responsibilities	**Collaborations**
This class is an interface for generating files.	-

Class Name: InfoWriter	
Responsibilities	**Collaborations**
This class is responsibly for writing to the taxpayer's _INFO files and delegates Receipt methods.	FileWriter

Class Name: TXTInfoWriter

Responsibilities	Collaborations
This class provides simple methods for writing to the txt _INFO files.	InfoWriter

Class Name: XMLInfoWriter	
Responsibilities	**Collaborations**
This class provides simple methods for writing to the xml _INFO files.	InfoWriter

Class Name: InfoWriterFactory	
Responsibilities	**Collaborations**
This class creates a specific InfoWriter object.	FileWriter TXTInfoWriter XMLInfoWriter

Class Name: LogWriter	
Responsibilities	**Collaborations**
This class is responsibly for keeping log files of a taxpayer.	FileWriter

Class Name: TXTLogWriter	
Responsibilities	**Collaborations**
This class provides simple methods for writing to the txt log files.	LogWriter

Class Name: XMLLogWriter	
Responsibilities	**Collaborations**
This class provides simple methods for writing to the xml log files.	LogWriter

Class Name: LogWriterFactory	
Responsibilities	**Responsibilities**
This class creates a specific LogWriter object.	FileWriter TXTLogWriter XMLLogWriter

Class Name: Taxpayer	
Responsibilities	**Responsibilities**
This class represents the main attributes and functions of a taxpayer.	Receipt

Class Name: SingleTaxpayer	
Responsibilities	**Responsibilities**
This class initializes the fields of its parent class Taxpayer with constants of a single taxpayer.	Taxpayer

Class Name: MarriedFilingSeperatelyTaxpayer	
Responsibilities	**Responsibilities**
This class initializes the fields of its parent class Taxpayer with constants of a Married Filing Separately taxpayer.	Taxpayer

Class Name: MarriedFilingJointlyTaxpayer	
Responsibilities	**Responsibilities**

This class initializes the fields of its parent class Taxpayer with constants of a Married Filing Jointly taxpayer.	Taxpayer

Class Name: HeadOfHouseholdTaxpayer	
Responsibilities	**Responsibilities**
This class initializes the fields of its parent class Taxpayer with constants of a Head Of Household taxpayer.	Taxpayer

Class Name: TaxpayerFactory	
Responsibilities	**Responsibilities**
This class creates a specific Taxpayer object.	Taxpayer and all its subclasses

Class Name: TaxpayerManager	
Responsibilities	**Responsibilities**
This class is responsibly for data holding of taxpayers and provides the main use cases.	Receipt Taxpayer

	TaxpayerFactory Company

Class Name: Receipt	
Responsibilities	**Responsibilities**
This class represents a receipt.	Date Company

Class Name: Date	
Responsibilities	**Responsibilities**
This class is used by Receipt class to know when a receipt was given.	-

Class Name: Company	
Responsibilities	**Responsibilities**
This class is used by Receipt class to represent the company that gave a receipt	Address

Class Name: Address	
Responsibilities	**Responsibilities**
This class is used by Company class to keep basic info about its address.	-

Class Name: All classes ending with Exception	
Responsibilities	**Responsibilities**
These Classes are used to provide warning messages to users when they do something that may risk the stable state of the system.	-

Class Name: ChartDisplay	
Responsibilities	**Responsibilities**
This class creates bar chart and pie chart where user can have a graphical representation of a taxpayer’s tax analysis.	-

Class Name: TaxpayerData	
Responsibilities	**Responsibilities**

This class provides the UI of a taxpayer’s main function(add receipt, remove receipt etc.)	ChartDisplay TaxpayerManager Receipt

Class Name: GraphicalInterface	
Responsibilities	**Responsibilities**
This class is the app’s graphical interface.	TaxpayerData TaxpayerManager