MINESSOTA INCOME TAX CALCULATOR

REENGINEERING THE LEGACY CODE

OVERALL REPORT

VERSION <1.0>

Varsos Vasileios | AM:4323

TABLE OF CONTENTS

Introduction 4

Refactored Design 4

- Use Cases 4
- Architecture 8
- Detailed Design 8
- Classes Responcibilities and Collaborations (CRC CARDS) 11

INTRODUCTION

The goal of the project was to reengineer a Java application. We were given an already completed project that was an income tax calculator application for Minnesota state citizens. The tax calculation occurred using receipts uploaded to the system, as long as other necessary information for each citizen using txt or xml files as input. Our main requirements were to understand the legacy code in depth and via the use of refactoring methods to improve the quality of the code.

REFACTORED DESIGN

USE CASES

Use case ID	UC1
Actors	The user
Pre conditions	None
Main flow of events	1. The use case starts when the user clicks on the "Load Taxpayer" function 2. The system allows the user to insert the tax registration number and select format 3. The system loads the taxpayer's information using a file of the selected format and TRN
Alternative flow 1	If the user inserts a tax registration number that is not 9 digits the system prompts the user to try again
Post conditions	The taxpayer is loaded in the system and displayed to the user

Use case ID	<u>UC2</u>
Actors	<u>The user</u>
Pre conditions	<u>None</u>
Main flow of events	1. The use case starts when the user selects the "Select Taxpayer" function 2. The system prompts the user to insert the tax registration number of the taxpayer the user wishes to select
Alternative flow 1	If the taxpayer is not loaded into the system, the user is prompted to try again
Alternative flow 2	If the user inserts a taxpayer number that is not 9 digits the user is prompted to try again
Post conditions	The system provides the information window of the selected taxpayer

Use case ID	<u>UC3</u>
Actors	<u>The user</u>
Pre conditions	<u>A taxpayer is selected using the "Select Taxpayer" function (UC2)</u>
Main flow of events	1. The use case starts when the user selects the "Add Receipt" function 2. The system prompts the user to insert the receipt information
Alternative flow 1	If the user provides a receipt kind that does not correspond the available options, they are prompted to try again
Post conditions	The receipt is loaded into the system

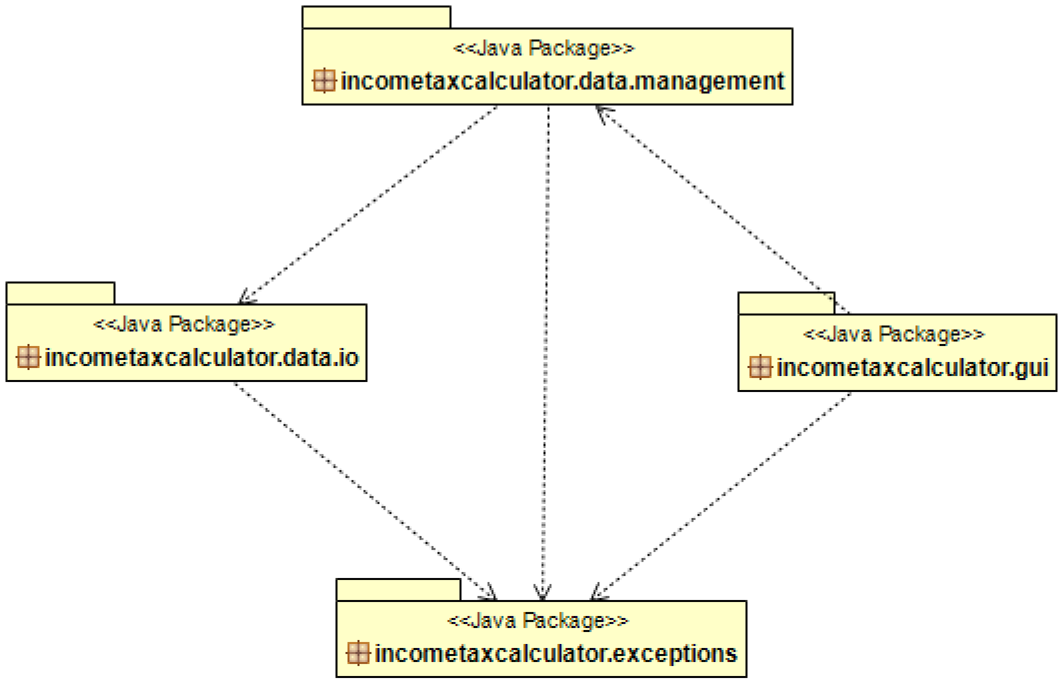
Use case ID	UC4
Actors	The user
Pre conditions	A taxpayer is selected using the “Select Taxpayer” function (UC2)
Main flow of events	1. The use case starts when the user selects the “Delete Receipt” function 2. The system prompts the user to insert the Receipt ID
Post conditions	The receipt is deleted from the system

Use case ID	UC5
Actors	The user
Pre conditions	A taxpayer is selected using the “Select Taxpayer” function (UC2)
Main flow of events	3. The use case starts when the user selects the “View Report” function 4. The system prints out a report of the taxpayer containing: 4.1. A bar chart containing the basic tax plus the added tax of the selected taxpayer 4.2. A pie chart analyzing the value of the submitted receipts for each category
Post conditions	The report is printed out to the user

Use case ID	UC6
Actors	The user
Pre conditions	A taxpayer is selected using the “Select Taxpayer” function (UC2)
Main flow of events	1. The use case starts when the user selects the “Save Data” function 2. The system prompts the user to select a format for the saved file
Post conditions	The taxpayer report is saved in the selected file format

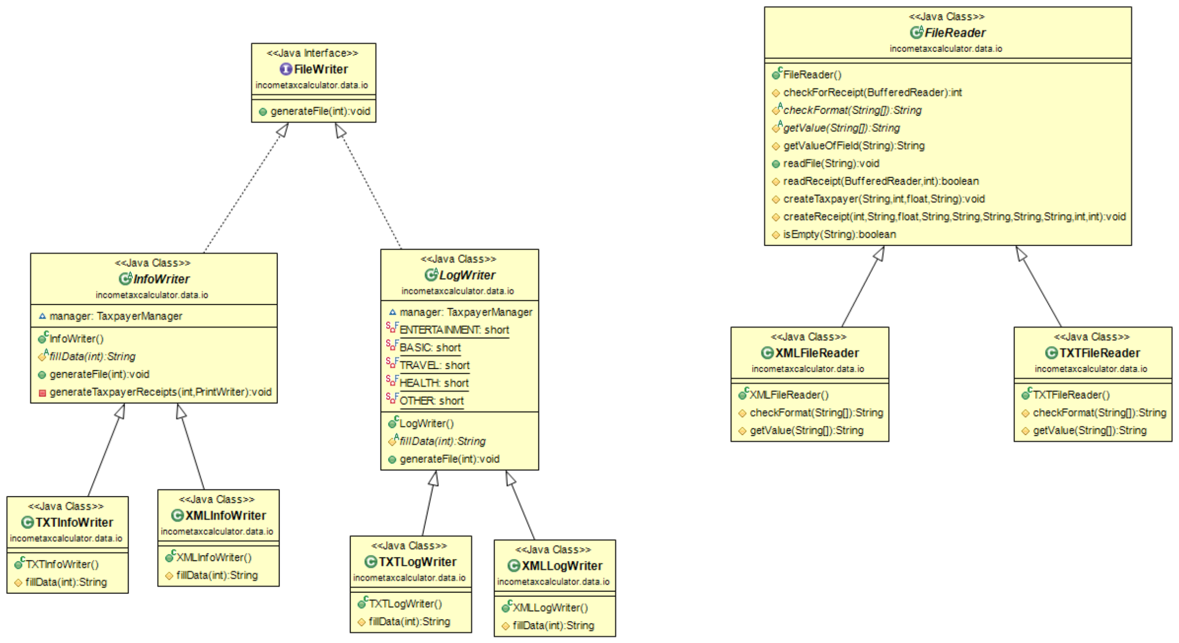
Use case ID	UC7
Actors	The user
Pre conditions	None
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 3. The use case starts when the user selects the “Delete Taxpayer” function 4. The system prompts the user to insert a tax registration number
Post conditions	The selected taxpayer is deleted from the system

ARCHITECTURE

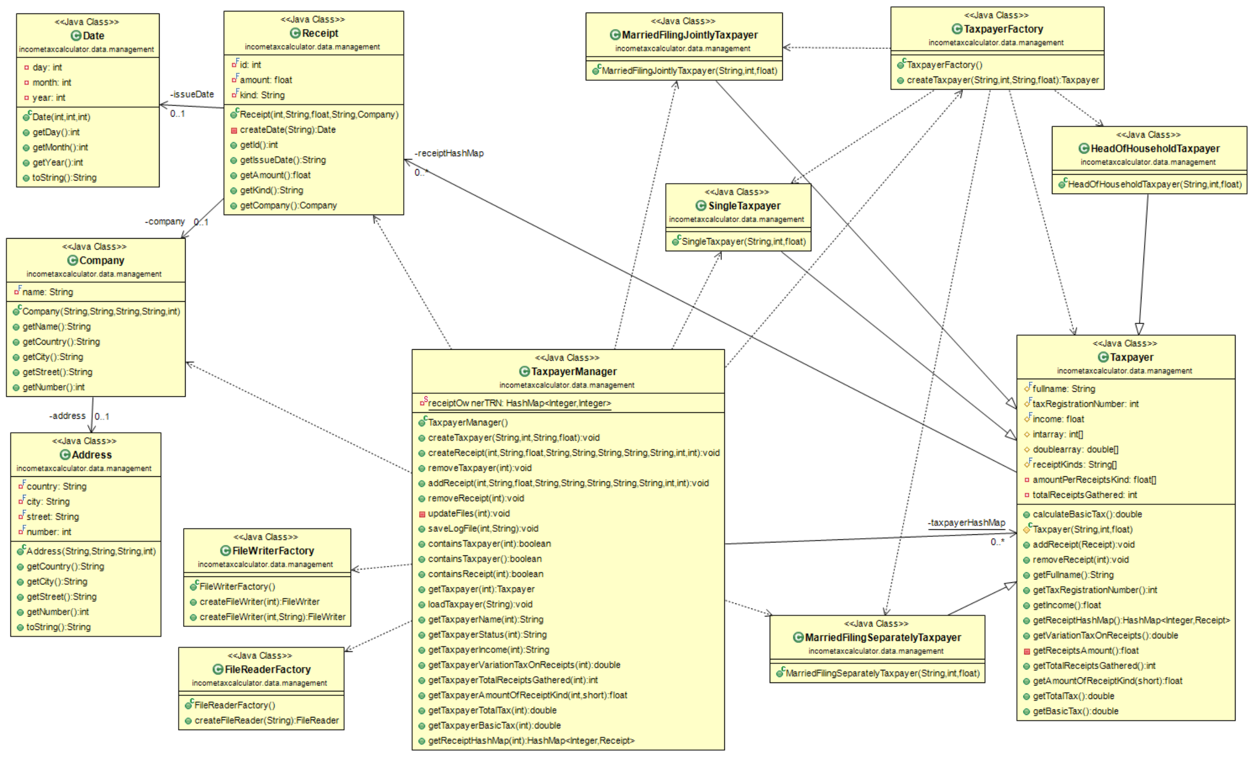

<<Java Package>>
incometaxcalculator.data.management
<<Java Package>>
incometaxcalculator.data.io
<<Java Package>>
incometaxcalculator.gui
<<Java Package>>
incometaxcalculator.exceptions

DETAILED DESIGN

- <u>Incometaxcalculator.data.io</u>



- Incometaxcalculator.data.management



<<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>>
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
-issueDate
0..1
-company
0..1
-receiptHashMap
0..*
<<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
<<Java Class>>
MarriedFilingJointlyTaxpayer
incometaxcalculator.data.management
MarriedFilingJointlyTaxpayer(String,int,float)
<<Java Class>>
TaxpayerFactory
incometaxcalculator.data.management
TaxpayerFactory()
createTaxpayer(String,int,String,float):Taxpayer
<<Java Class>>
HeadOfHouseholdTaxpayer
incometaxcalculator.data.management
HeadOfHouseholdTaxpayer(String,int,float)
<<Java Class>>
SingleTaxpayer
incometaxcalculator.data.management
SingleTaxpayer(String,int,float)
<<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>>
Taxpayer
incometaxcalculator.data.management
fullname: String
taxRegistrationNumber: int
income: float
intarray: int[]
doublearray: double[]
receiptKinds: String[]
amountPerReceiptsKind: float[]
totalReceiptsGathered: int
calculateBasicTax():double
Taxpayer(String,int,float)
addReceipt(Receipt):void
removeReceipt(int):void
getFullname():String
getTaxRegistrationNumber():int
getIncome():float
getReceiptHashMap():HashMap<Integer,Receipt>
getVariationTaxOnReceipts():double
getReceiptsAmount():float
getTotalReceiptsGathered():int
getAmountOfReceiptKind(short):float
getTotalTax():double
getBasicTax():double
-taxpayerHashMap
0..*
<<Java Class>>
FileWriterFactory
incometaxcalculator.data.management
FileWriterFactory()
createFileWriter(int):FileWriter
createFileWriter(int,String):FileWriter
<<Java Class>>
FileReaderFactory
incometaxcalculator.data.management
FileReaderFactory()
createFileReader(String):FileReader
<<Java Class>>
MarriedFilingSeparatelyTaxpayer
incometaxcalculator.data.management
MarriedFilingSeparatelyTaxpayer(String,int,float)

The problems in the initial design were addressed using the guidance of the Project backlog as well as various refactoring methods shown in class.

- Incometaxcalculator.data.io package

1. **Company Class**: The class contained an unnecessary method that did not contribute to the final project. ***The method was removed***.

2. **Taxpayer Class:** To combat the unnecessary complexity in the methods described, I created an array of Strings with the various kinds of receipts, and simplified the conditional logic by iterating through the array instead of having different if-statements for each one. In the getVariationTaxOnReceipts() method, there was also an additional method created to return the total amount of receipts, instead of calculating it in the method itself, to further simplify the conditional logic.

3. **Subclasses of the Taxpayer class:** Instead of having different implementations of the calculateBasicTax() method, I used parametrization to create a template in the Taxpayer class and stored the separate values in two different arrays (for integers and doubles). The subclasses instead just initialized the parameters inside their constructors.

4. **Taxpayer manager class:** This class was a concrete example of a "God class", with many responsibilities. I delegated some of its responsibilities to subordinate classes by creating the classes TaxpayerFactory(), FileReaderFactory() and FileWriterFactory() that served as parametrized factories for their respective object types. In the case of FileWriter objects, as there could be two different variations that required different parameters (either for txt/xml or log files), I implemented two different constructors that return either InfoWriter or LogWriter objects depending on the parameters.

- Incometaxcalculator.data.io package

1. **TXT/XML FileReader Classes:** The classes were subject to a lot of duplicate code. Using **Extract Class** method, I pulled up the main concepts of to the FileReader Class, and instead implemented the abstract methods checkFormat(), that runs the conditional logic that returns the file format(txt/xml) in the checkForReceipt() method, and getValue(), which runs the main conditional logic that was different for the getValueOfField() method.

2. **FileWriter Class:** This class was a **Middle Man** for TaxpayerManager() class, all of the delegation was removed and instead moved over to the methods needed in the package. Instead, the FileWriter Class was turned into an interface implementing the generateFile() method.

3. **TXT/XML InfoWriter Classes:** The classes contained a lot of duplicate code. I created a new abstract InfoWriter class that contained parametrized variants of the classes' methods. In return, the subclasses implemented the abstract method fillData, that return various string constants.

4. **TXT/XML LogWriter Classes:** Similar to the Infowriter class, I created a LogWriter super class that implemented the FileWriter interface and abstracted the core ideas of the classes' algorithms. The subclasses returned string constants that initialized the parameters.

CLASSES RESPONSIBILITIES AND COLLABORATIONS (CRC CARDS)

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

Class Name: Date	
Responsibilities	**Collaborations**
▪ This class is responsible for returning information about a Receipt's Date	▪ This class is associated with the class Receipt()

Class Name: Address	
Responsibilities	**Collaborations**
▪ This class is responsible for returning information about the Address of a Company	▪ This class is associated with the class Company()

Class Name: Company	
Responsibilities	**Collaborations**
▪ This class is responsible for returning information about a Company listed in a Receipt	▪ This class is associated with the classes Receipt() and Address, and a dependent of TaxpayerManager()

Class Name: Receipt	
Responsibilities	**Collaborations**
▪ This class is responsible for creating and storing various information regarding Receipts	▪ This class is a dependent of the classTaxpayerManager() and associated with the classes Date(), Taxpayer() and Company()

Class Name: MarriedFillingSeparately	
Responsibilities	**Collaborations**
▪ This class is responsible for instantiating MarriedFilllingSeparately type of Taxpayer objects	▪ This class is a dependent of the classes TaxpayerManager() and TaxpayerFactory() and dependent upon the class Taxpayer()

Class Name: Taxpayer	
Responsibilities	**Collaborations**
▪ This class is responsible for containing information about individual Taxpayer type of objects, such as their personal info, receipt info, as well as tax calculations	▪ This class is associated with the class es Receipt(), HeadOfHouseholdTaxpayer(),TaxpayerFactory(), MarriedFillingJointlyTaxpayer(),SingleTaxpayer() and MarriedFillingSeperatelyTaxpayer()

Class Name: MarriedFillingJointly	
Responsibilities	**Collaborations**
▪ This class is responsible for instantiating MarriedFilllingJointly type of Taxpayer objects	▪ This class is a dependent of the classes TaxpayerManager() and TaxpayerFactory() and associated with the class Taxpayer()

Class Name: HeadOfHousehold	
Responsibilities	**Collaborations**
▪ This class is responsible for instantiating HeadOfHousehold type of Taxpayer objects	▪ This class is a dependent of the class TaxpayerFactory() and associated with the class Taxpayer()

Class Name: SingleTaxpayer	
Responsibilities	**Collaborations**
▪ This class is responsible for instantiating SingleTaxpayer type of Taxpayer objects	▪ This class is a dependent of the classes TaxpayerManager() and TaxpayerFactory() and associated with the class Taxpayer()

Class Name: TaxpayerFactory	
Responsibilities	**Collaborations**
▪ This class is responsible for instantiating Taxpayer objects	▪ This class is a dependent of the class TaxpayerManager() and dependent upon the class Taxpayer(), as well as all of the Taxpayer subclasses

Class Name: TaxpayerManager	
Responsibilities	**Collaborations**
▪ This class is responsible for managing the Taxpayer and Receipts database, delegating various methods in regards to creating and maintaining the data (hashmap)	▪ This class is dependent upon the Taxpayer() subclasses as well as the TaxpayerFactory(), Receipts(),Company(),FileWriterFactory and FileReaderFactory() classes and associated with the Taxpayer class()

Class Name: FileReaderFactory	
Responsibilities	**Collaborations**
▪ This class is responsible for instantiating FileReader objects based on the file format	▪ This class is a dependent of the class TaxpayerManager()

Class Name: FileWriterFactory	
Responsibilities	**Collaborations**
▪ This class is responsible for instantiating FileWriter info type or log type objects based on initialization of the method	▪ This class is a dependent of the class TaxpayerManager()

Class Name: InfoWriter	
Responsibilities	**Collaborations**
▪ This class is responsible for displaying information about Taxpayers and Receipts based on file format	▪ This class is a associated with the classes TXTInfoWriter and XMLInfoWriter and depended upon the FileWriter interface

Class Name: LogWriter	
Responsibilities	**Collaborations**
▪ This class is responsible for displaying information about Taxpayers and Receipts based on file format	▪ This class is a associated with the classes TXTLogWriter and XMLLogWriter and depended upon the FileWriter interface

Class Name: TXTInfoWriter	
Responsibilities	**Collaborations**
▪ This class is responsible for instantiating the txt type of InfoWriter objects in proper format	▪ This class is a associated with the class InfoWriter

Class Name: XMLInfoWriter	
Responsibilities	**Collaborations**
▪ This class is responsible for instantiating the xml type of InfoWriter objects in proper format	▪ This class is a associated with the class InfoWriter

Class Name: TXTLogWriter	
Responsibilities	**Collaborations**
▪ This class is responsible for instantiating the txt type of LogWriter objects in proper format	▪ This class is a associated with the class LogWriter

Class Name: XMLLogWriter	
Responsibilities	**Collaborations**
▪ This class is responsible for instantiating the xml type of LogWriter objects in proper format	▪ This class is a associated with the class LogWriter

Class Name: XMLFileReader	
Responsibilities	**Collaborations**
▪ This class is responsible for recognizing and parsing xml type of documents	▪ This class is a associated with the class FileReader

Class Name: TXTFileReader	
Responsibilities	**Collaborations**
▪ This class is responsible for recognizing and parsing txt type of documents	▪ This class is a associated with the class FileReader

Class Name: FileReader	
Responsibilities	**Collaborations**
▪ This class is responsible for parsing input documents and storing the values to create Taxpayer or Receipt type objects	▪ This class is a associated with the classes TXTFileReader and XMLFileReader

Interface Name: FileWriter	
Responsibilities	**Collaborations**
▪ This class is responsible for generating the FileWriter type objects	▪ This class is a associated with the classes TXTFileWriter and XMLFileWriter

Class Name: WrongTaxpayerStatusException	
Responsibilities	**Collaborations**
▪ This class is an exception class responsible for catching errors in the taxpayer status while parsing an input document	▪ This class is dependent upon the TaxpayerManager and FileReader/Taxpayer Factory classes

Class Name: WrongFileEndingException	
Responsibilities	**Collaborations**
▪ This class is an exception class responsible for catching errors in the file ending while parsing an input document	▪ This class is dependent upon the TaxpayerManager and FileReaderFactory classes

Class Name: WrongFileFormatException	
Responsibilities	**Collaborations**
▪ This class is an exception class responsible for catching errors in the file format while parsing an input document	▪ This class is dependent upon the TaxpayerManager, FileWriterFactory and FileReaderFactory classes

Class Name: ReceiptAlreadyExists	
Responsibilities	**Collaborations**
▪ This class is an exception class responsible for catching errors in the createReceipt method in the case of a Duplicate receipt	▪ This class is dependent upon the TaxpayerManager class

Class Name: WrongReceiptKindException	
Responsibilities	**Collaborations**
▪ This class is an exception class responsible for catching errors in the Receipt kind upon parsing an input document	▪ This class is dependent upon the TaxpayerManager and FileReaderFactory classes

Class Name: WrongReceiptDateException	
Responsibilities	**Collaborations**
▪ This class is an exception class responsible for catching errors in the Receipt date upon parsing an input document	▪ This class is dependent upon the TaxpayerManager, Receipt and FileReaderFactory classes