MINESSOTA INCOME TAX CALCULATOR

OVERALL REPORT

VERSION <1.0>

Sofia Massara - 4729

Despoina-Markela Papageorgiou - 4893

TABLE OF CONTENTS

INTRODUCTION

The objective of this project is to calculate the taxes of the citizens of Minnesota state, USA. The “Taxpayers” are categorized as “Married Filing Separately”, “Married Filing Jointly”, “Head of Household” and “Single”. For each category, the basic tax is calculated separately. This project is also accompanied by a GUI application, making it possible for the user to view and edit each Taxpayer’s information.

REFACTORED DESIGN

USE CASES

Use case ID	UC1: Load Information
Actors	Application User
Pre conditions	The <AFM>_INFO.txt (or .xml) from the taxpayer, needs to be already loaded in the system.
Main flow of events	1. The use case starts when the user selects the specific taxpayers file.
Alternative flow 1	(none)
Post conditions	The taxpayers information is loaded.

Use case ID	UC2: Add Receipt
Actors	User
Pre conditions	(none)
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the user chooses a taxpayer. 2. Fill in receipt information: Id, date, amount etc.
Alternative flow 1	In case of false receipt information or the receipt already being loaded an exception will be thrown.
Post conditions	The <AFM>_INFO.txt files contents are renewed.

Use case ID	UC3: Delete Receipts
Actors	User
Pre conditions	The taxpayer has submitted at least one receipt
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the user chooses a taxpayer.
Alternative flow 1	If no receipts are loaded, then then the use case can’t proceed.
Post conditions	The <AFM>_INFO.txt files contents are renewed.

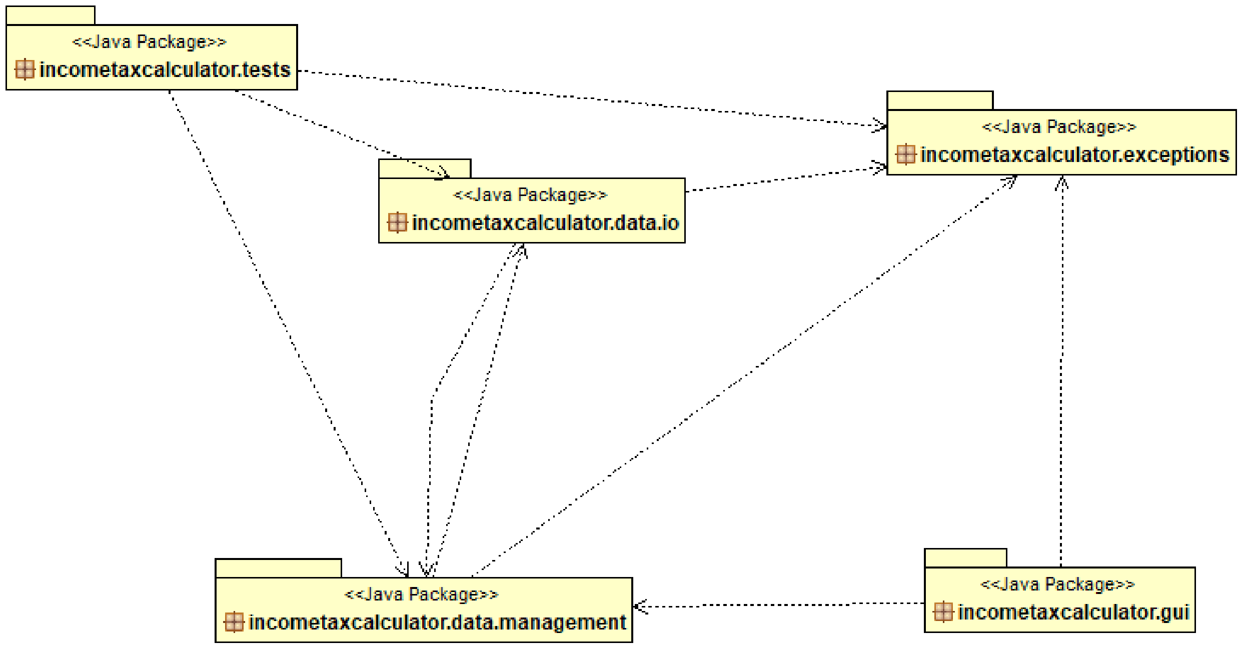
Use case ID	UC4: Calculate Report
Actors	User
Preconditions	Taxpayers’ information need to be loaded
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the user chooses a taxpayer. 2. The user chooses chart type: Bar chart or Pie chart 3. The user fills in the information needed for each chart type 3.1. For bar chart: 3.2. For pie chart:
Postconditions	Show report

Use case ID	UC5: Save Information
Actors	User
Pre conditions	Taxpayers’ information needs to be loaded
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the user chooses a taxpayer. 2. The user chooses the format of the file (TXT or XML)
Post conditions	The output is <AFM>_LOG.txt (or.xml) file.

Use case ID	UC6: Remove Taxpayer
Actors	User
Preconditions	The taxpayer is added to the list
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the user chooses a taxpayer to remove.
Post conditions	The chosen taxpayer is removed

ARCHITECTURE

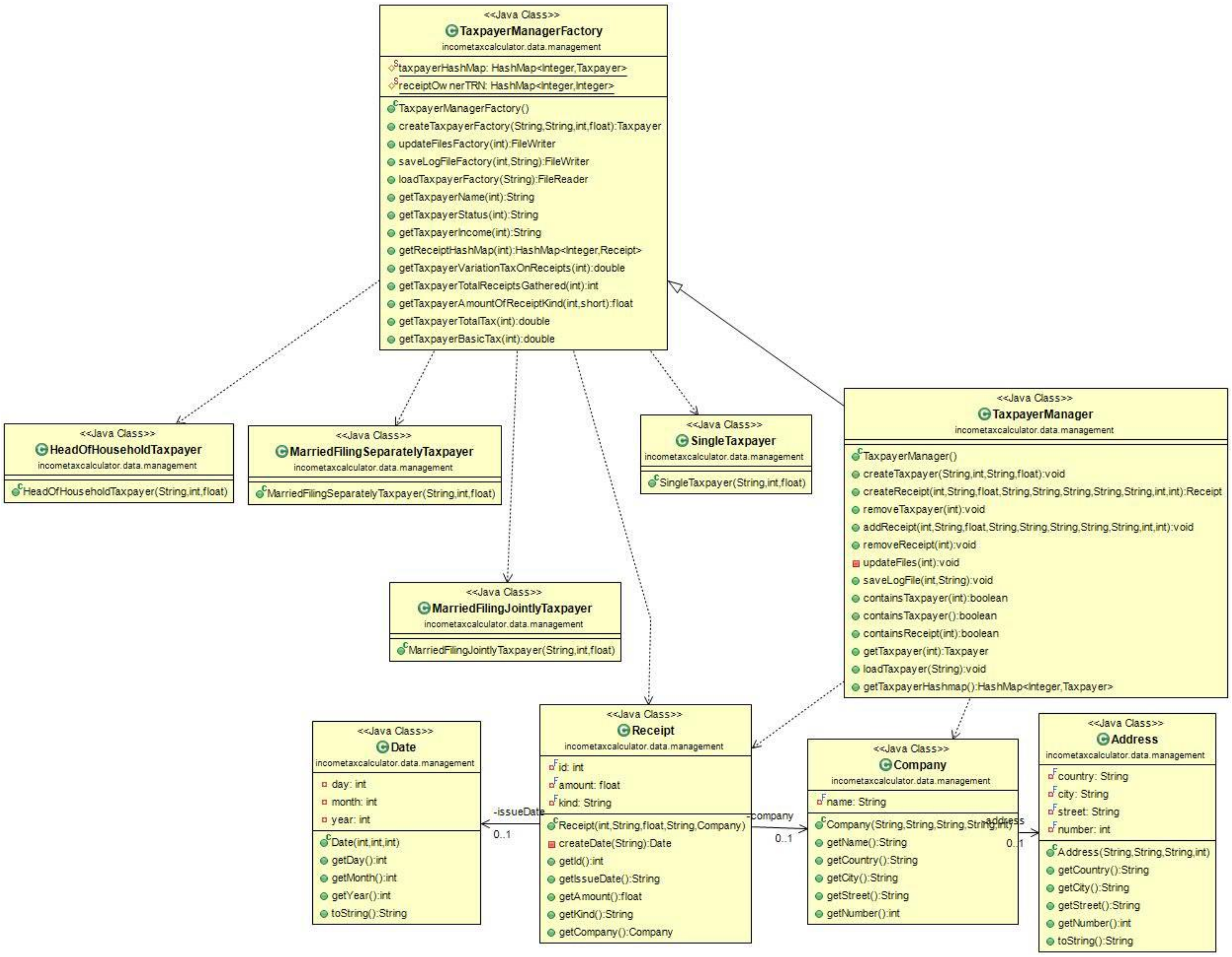
- UML package diagram:



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

DETAILED DESIGN

- incometaxcalculator.data.management package:



<<Java Class>>
TaxpayerManagerFactory
incometaxcalculator.data.management
taxpayerHashMap: HashMap<Integer,Taxpayer>
receiptOwnerTRN: HashMap<Integer,Integer>
TaxpayerManagerFactory()
createTaxpayerFactory(String,String,int,float):Taxpayer
updateFilesFactory(int):FileWriter
saveLogFileFactory(int,String):FileWriter
loadTaxpayerFactory(String):FileReader
getTaxpayerName(int):String
getTaxpayerStatus(int):String
getTaxpayerIncome(int):String
getReceiptHashMap(int):HashMap<Integer,Receipt>
getTaxpayerVariationTaxOnReceipts(int):double
getTaxpayerTotalReceiptsGathered(int):int
getTaxpayerAmountOfReceiptKind(int,short):float
getTaxpayerTotalTax(int):double
getTaxpayerBasicTax(int):double
<<Java Class>>
HeadOfHouseholdTaxpayer
incometaxcalculator.data.management
HeadOfHouseholdTaxpayer(String,int,float)
<<Java Class>>
MarriedFilingSeparatelyTaxpayer
incometaxcalculator.data.management
MarriedFilingSeparatelyTaxpayer(String,int,float)
<<Java Class>>
SingleTaxpayer
incometaxcalculator.data.management
SingleTaxpayer(String,int,float)
<<Java Class>>
TaxpayerManager
incometaxcalculator.data.management
TaxpayerManager()
createTaxpayer(String,int,String,float):void
createReceipt(int,String,float,String,String,String,String,String,int,int):Receipt
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
getTaxpayerHashmap():HashMap<Integer,Taxpayer>
<<Java Class>>
MarriedFilingJointlyTaxpayer
incometaxcalculator.data.management
MarriedFilingJointlyTaxpayer(String,int,float)
<<Java Class>>
Date
incometaxcalculator.data.management
day: int
month: int
year: int
Date(int,int,int)
getDay():int
getMonth():int
getYear():int
toString():String
-issueDate
0..1
<<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
-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

Picture1: UML class diagram for incometaxcalculator.data.management package.

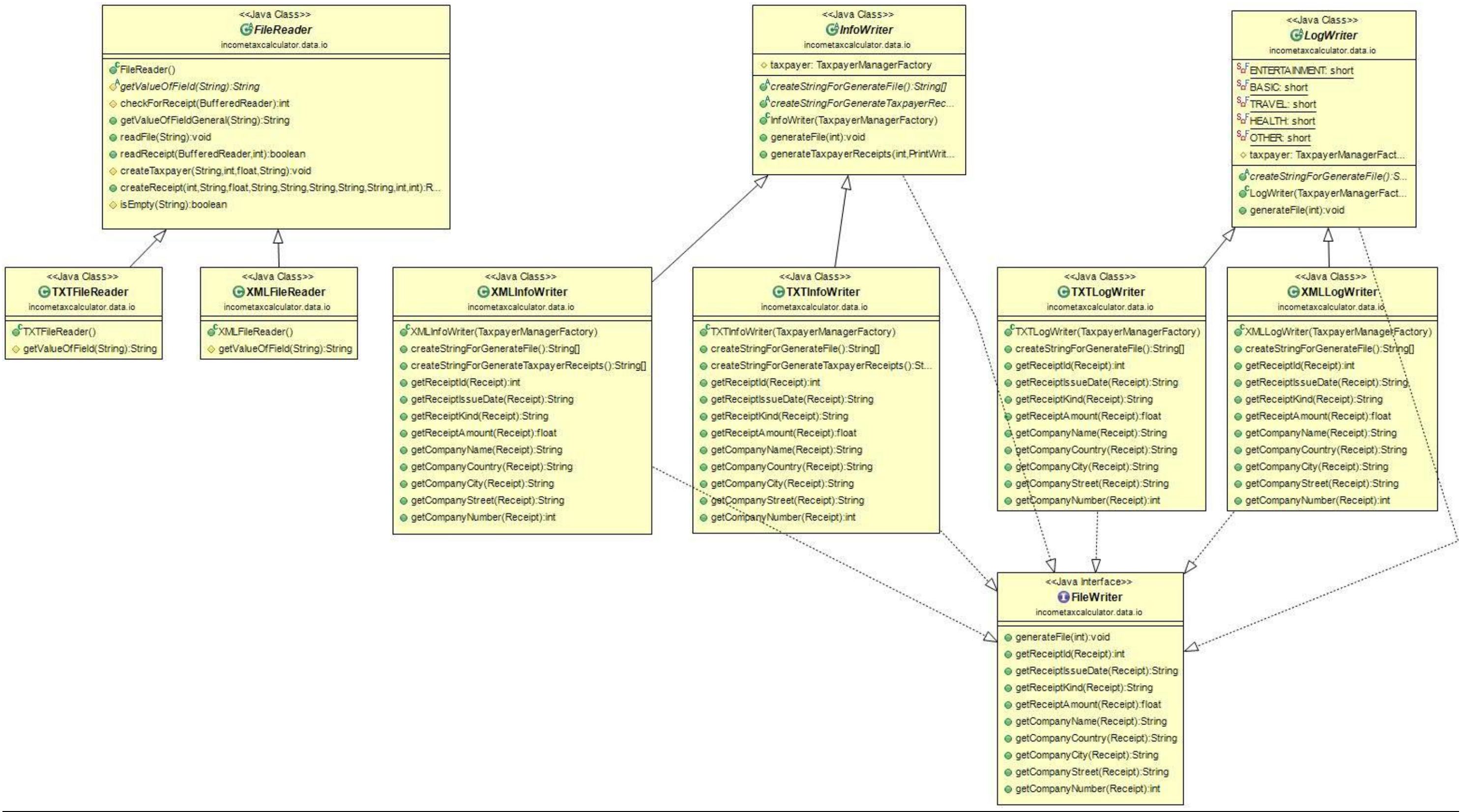
1. We removed the getAddress() method , as it was dead code.

2. To replace complex conditional logic in the form of chained if-else statements, we created the array Lists receiptKind, incomeCompare and taxReturn (containing the necessary variables) and implemented them inside a for loop.

3. To deal with duplicate code in the Taxpayer subclasses, we moved the method calculateBasicTax() in the super class Taxpayer and used simple arrays as fields ,which were initialized with the needed constants in each subclass constructor. Then we modified the method calculateBasicTax() in order to use these arrays instead of constants.

4. In this case, we created a simple parameterized factory, TaxpayerManagerFactory ,that creates different types of objects needed in each method : createTaxpayer(), updateFiles(), saveLogFile() and loadTaxpayer() . Those factory methods also contain a simplified version of the previous complex if-else and conditional logic.

- “incometaxcalculator.data.io”package **:**



<<Java Class>>
FileReader
incometaxcalculator.data.io
FileReader()
getValueOfField(String):String
checkForReceipt(BufferedReader):int
getValueOfFieldGeneral(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):R...
isEmpty(String):boolean
<<Java Class>>
InfoWriter
incometaxcalculator.data.io
taxpayer: TaxpayerManagerFactory
createStringForGenerateFile():String[]
createStringForGenerateTaxpayerRec...
InfoWriter(TaxpayerManagerFactory)
generateFile(int):void
generateTaxpayerReceipts(int,PrintWrit...
<<Java Class>>
LogWriter
incometaxcalculator.data.io
ENTERTAINMENT: short
BASIC: short
TRAVEL: short
HEALTH: short
OTHER: short
taxpayer: TaxpayerManagerFact...
createStringForGenerateFile():S...
LogWriter(TaxpayerManagerFact...
generateFile(int):void
<<Java Class>>
TXTFileReader
incometaxcalculator.data.io
TXTFileReader()
getValueOfField(String):String
<<Java Class>>
XMLFileReader
incometaxcalculator.data.io
XMLFileReader()
getValueOfField(String):String
<<Java Class>>
XMLInfoWriter
incometaxcalculator.data.io
XMLInfoWriter(TaxpayerManagerFactory)
createStringForGenerateFile():String[]
createStringForGenerateTaxpayerReceipts():String[]
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 Class>>
TXTInfoWriter
incometaxcalculator.data.io
TXTInfoWriter(TaxpayerManagerFactory)
createStringForGenerateFile():String[]
createStringForGenerateTaxpayerReceipts():St...
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 Class>>
TXTLogWriter
incometaxcalculator.data.io
TXTLogWriter(TaxpayerManagerFactory)
createStringForGenerateFile():String[]
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 Class>>
XMLLogWriter
incometaxcalculator.data.io
XMLLogWriter(TaxpayerManagerFactory)
createStringForGenerateFile():String[]
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
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

Picture2: UML class diagram for “incometaxcalculator.data.io” package.

1. The problem here was Duplicate Code. For TXTFileReader, XMLFileReader classes, we extracted the a similar code to the method getValueOfFieldGeneral(String fieldsLine) inside the abstract FileReader class, which is extended by them. In the TXTFileReader, XMLFileReader and classes there is only the necessary code inside the method getValueOfField(String fieldsLine).

2. By turning the FileWriter class into an interface instead of the abstract class, we simplified and reduced some methods. More specifically we removed some methods that were acting as a Middle Man for TaxPayerManager class and pushed them down to classes that implement the interface, some others that were only useful there.

3. To deal with duplicate code inside the TXTInfoWriter and XMLInfoWriter classes, we created the methods createStringForGenerateFile() ,createStringForGenerateTaxpayerReceipts() that simply provide constant string tags inside array Lists . These methods are used inside the abstract InfoWriter class where we pulled up the similar code for generatedFile() and generateTaxpayerReceipts() methods.

4. Similarly we dealt with duplicate code inside the TXTLogWriter and XMLLogWriter classes by creating the createStringForGenerateFile() method, to provide the constant string tags with an array List , which is later used inside the generateFile(int taxRegistrationNumber) method , in LogWriter abstact class.

CLASSES RESPONSIBILITIES AND COLLABORATIONS (CRC CARDS)

- A brief description for each class in terms of a CRC card:

Class Name: Abstract FileReader class	
Responsibilities	**Collaborations**
• This class is responsible for reading files and receipts • This class created Taxpayers and receipts	• It is a super class for XMLFilerReader and TXTFileReader . • Creates Taxpayers and receipts through TaxpayerManager objects.

Class Name: interface FileWriter class	
Responsibilities	**Collaborations**
• This interface provides some necessary methods for its subclasses	• It is implemented by InfoWriter , LogWriter, TXTInfoWriter,XMLInfoWriter,TXTLogWriter , XMLLogWriter

Class Name: abstract InfoWriter class	
Responsibilities	**Collaborations**
• This class is responsible for generating Info files.	• This class is extended by TXTInfoWriter and XMLInfoWriter.

Class Name: abstract LogWriter class	
Responsibilities	**Collaborations**
• This class is responsible for generating Log files.	• This class is extended by TXTLogWriter and XMLLogWriter.

Class Name: Address class	
Responsibilities	**Collaborations**
• This class provides the necessary fields for an Address object	(none)

Class Name: Company class	
Responsibilities	**Collaborations**
• This class provides the necessary fields for a Company object	(none)

Class Name: Receipt class	
Responsibilities	**Collaborations**
• This class provides the necessary fields for a Receipt object. • This class is responsible for creating a date the needed way.	• This class creates a Date object and a Company object.

Class Name: abstract Taxpayer class	
Responsibilities	**Collaborations**
• This class is responsible for handling basic information about each Taxpayer. • This class is responsible for calculating tax for each type of Taxpayer.	• This class is extended by HeadOfHouseholdTaxpayer , MarriedFilingJointlyTaxpayer, SingleTaxpayer, MarriedFilingSeparatelyTaxpayer.

Class Name: HeadOfHouseholdTaxpayer class	
Responsibilities	**Collaborations**
• This class provides the necessary_tax calculation constants for this type of Taxpayer.	• This class extends Taxpayer class

Class Name: MarriedFilingJointlyTaxpayer class	
Responsibilities	**Collaborations**
• This class provides the necessary_tax calculation constants for this type of Taxpayer.	• This class extends Taxpayer class

Class Name: MarriedFilingSeparatelyTaxpayer class	
Responsibilities	**Collaborations**
• This class provides the necessary_tax calculation constants for this type of Taxpayer.	• This class extends Taxpayer class

Class Name: SingleTaxpayer class	
Responsibilities	**Collaborations**
• Provides the necessary tax calculation constants for this type of Taxpayer.	• This class extends Taxpayer class

Class Name: TaxpayerManager class	
Responsibilities	**Collaborations**
• This class performs basic functions for each Taxpayer, Receipt, File. Like creating or removing. • This class contains Boolean functions to repetition of Taxpayers and Receipts.	• This class extends the TaxpayerManagerFactory class. • This class creates Taxpayer and Receipt objects.

Class Name: TaxpayerManagerFactory class	
Responsibilities	**Collaborations**
• This class contains methods that creates objects for each respective method in TaxpayerManager that needs them.	• It is extended by TaxpayerManager.

Class Name: FileTestTXT test class	
Responsibilities	**Collaborations**
• It is responsible for testing the main methods for creating and reading TXT type files.	(none)

Class Name: FileTestXML test class	
Responsibilities	**Collaborations**
• It is responsible for testing the main methods for creating and reading XML type files.	(none)

Class Name: TaxpayerManager test class	
Responsibilities	**Collaborations**
• This test is responsible for testing the methods concerning receipts and taxpayers and some exceptions.	(none)

IMPROVEMENT TASK

In the new, user-friendly, version of the GUI of the application, the user loads a taxpayer by clicking on the Load Taxpayer button, browsing through their files and then selecting the needed _INFO file, instead of typing the taxpayers AFM. If a _LOG file is selected an error message will appear. Also, the improved application doesn't contain a Select Taxpayer button, instead, that function can be performed by double-clicking on the loaded taxpayer from the list, on the Tax Registration Number window.