REENGINEERING OF MINESSOTA’S INCOME TAX CALCULATOR

OVERALL REPORT

VERSION 2.0

Kapenis Alexandros	AM: 2713
Parzigkas Christos	AM: 3317

TABLE OF CONTENTS

INTRODUCTION

The objective of this phase of the project is to reengineer the Minnesota Income Tax Calculation application. The application is used to calculate the income tax of Minnesota state citizens based on their marital status, income, and receipts. It also produces graphical representations of the data and generates output reports in txt or xml format. The goal of the reengineering process was to improve the overall structure and maintainability of the application, as well as to fix specific problems or issues with the legacy code.

To achieve these goals, we used a variety of techniques, including reading and understanding the documentation and source code and capturing the legacy architecture in UML diagrams. In addition, we used refactoring techniques such as extracting common code, simplifying complex methods, and using parameterized factories to delegate responsibilities to subordinate classes. Finally, we implement tests to ensure that our changes did not break any existing functionality.

REFACTORED DESIGN

USE CASES

- <u>UC01 – Load Taxpayer</u>

Use case ID	UC01
Actors	Tax Accountant
Preconditions	The taxpayer's file exists and is accessible to the user.
Main flow of events	1. The use case starts when the Tax Accountant selects to load taxpayer's information in the system. 2. The Tax Accountant inputs the taxpayer's AFM and selects the filetype (.txt or .xml). 3. The system retrieves the taxpayer's file.
Alternative flow 1	If the file is already loaded, the system displays an error message to the Tax Accountant.
Alternative flow 2	If the file doesn't exist, the system displays an error message to the Tax Accountant.
Post conditions	The system displays the taxpayer's AFM in a list of taxpayers AFMs.

➢ <u>UC02 – Select Taxpayer</u>

Use case ID	UC02
Actors	Tax Accountant
Preconditions	The system has at least one taxpayer’s file loaded.
Main flow of events	1. The use case starts when the Tax Accountant selects one taxpayer and presses the select button. 2. The system displays the selected taxpayer's details.
Alternative flow 1	If the list of taxpayers is empty, the system displays an error message.
Alternative flow 2	If the user hasn’t selected a taxpayer, the system displays an error message.
Post conditions	The system displays the selected taxpayer's details.

➢ <u>UC03 – Delete Taxpayer</u>

Use case ID	UC03
Actors	Tax Accountant
Preconditions	The system has at least one taxpayer’s file loaded.
Main flow of events	1. The use case starts when the Tax Accountant selects a taxpayer and hits the delete button. 2. The system displays a confirmation message. 3. The Tax Accountant confirms the deletion of the taxpayer. 4. The system deletes the taxpayer.

Alternative flow 1	If the Tax Accountant hit the cancel button, the system returns him to the list of taxpayers.
Post conditions	The taxpayer's information has been saved to a file.

➢ UC04 – Add New Receipt

Use case ID	UC04
Actors	Tax Accountant
Preconditions	A taxpayer has been selected and has his information displayed.
Main flow of events	1. The use case starts when the Tax Accountant selects to add a new receipt for the taxpayer. 2. The system displays a form to be filled. 3. The Tax Accountant enters the receipt details and hit the ok button. 4. The system updates the taxpayer's information with the new receipt. 5. The system updates the contents of the files with the updated taxpayer's information.
Alternative flow 1	If the Tax Accountant enters invalid receipt details, the system displays an error message.
Alternative flow 2	If the Tax Accountant hit the cancel button, the system returns him to the taxpayer's information screen.
Post conditions	The taxpayer's information is updated with the new receipt and the file is renewed with the updated information.

UC05 – Delete Receipt

Use case ID	UC05
Actors	Tax Accountant
Preconditions	1. A taxpayer has been selected and has his information displayed. 2. The taxpayer has at least one receipt loaded and displayed.
Main flow of events	1. The use case starts when the Tax Accountant selects a receipt and hit the delete button. 2. The system displays a confirmation message. 3. The Tax Accountant confirms the deletion of the receipt. 4. The system deletes the receipt and update the taxpayer's information. 5. The system updates the contents of the files with the updated taxpayer's information.
Alternative flow 1	If the Tax Accountant hit the cancel button, the system returns him to the taxpayer's information screen.
Post conditions	The taxpayer's information is updated with the deletion of the receipt and his file is renewed with the updated information.

UC06 – View Reports

Use case ID	UC06
Actors	Tax Accountant
Preconditions	A taxpayer has been selected and has his information displayed.
Main flow of events	1. The use case starts when the Tax Accountant selects to view a report for the taxpayer. 2. The system calculates the final tax, the basic tax and the increase or decrease of tax due to receipts. 3. The system displays a bar chart with the final tax, the basic tax and the increase

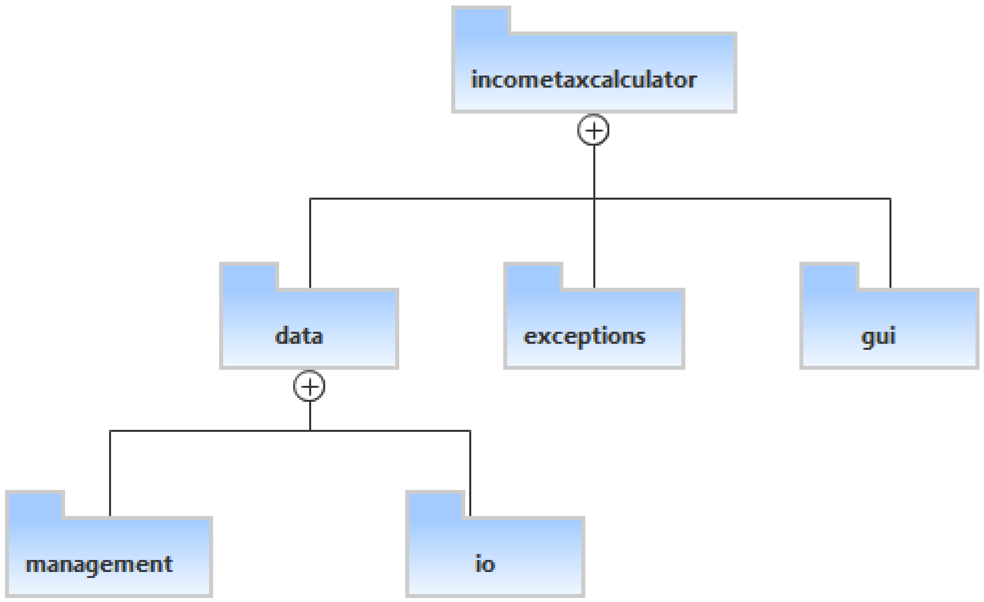
	or decrease of tax due to receipts. 4. The system calculates the total amount for each different receipt category. 5. The system displays a pie chart with the amount spent for each different receipt category.
Post conditions	The system displays a bar chart with the final tax, the basic tax and the increase or decrease of tax due to receipts. The system displays a pie chart with the total amount for each different receipt category.

➢ <u>UC07 – Save Taxpayer's Information</u>

Use case ID	UC07
Actors	Tax Accountant
Preconditions	A taxpayer has been selected and has his information displayed.
Main flow of events	1. The use case starts when the Tax Accountant selects to save the taxpayer's information in a file. 2. The system prompts the Tax Accountant to select the filetype, either .txt or .xml. 3. The user selects the file format and hit ok. 4. The system saves the file with same name as the taxpayer's AFM.
Alternative flow 1	If the Tax Accountant hit the cancel button, the system returns him to the taxpayer's information screen.
Post conditions	The taxpayer's information has been saved to a file.

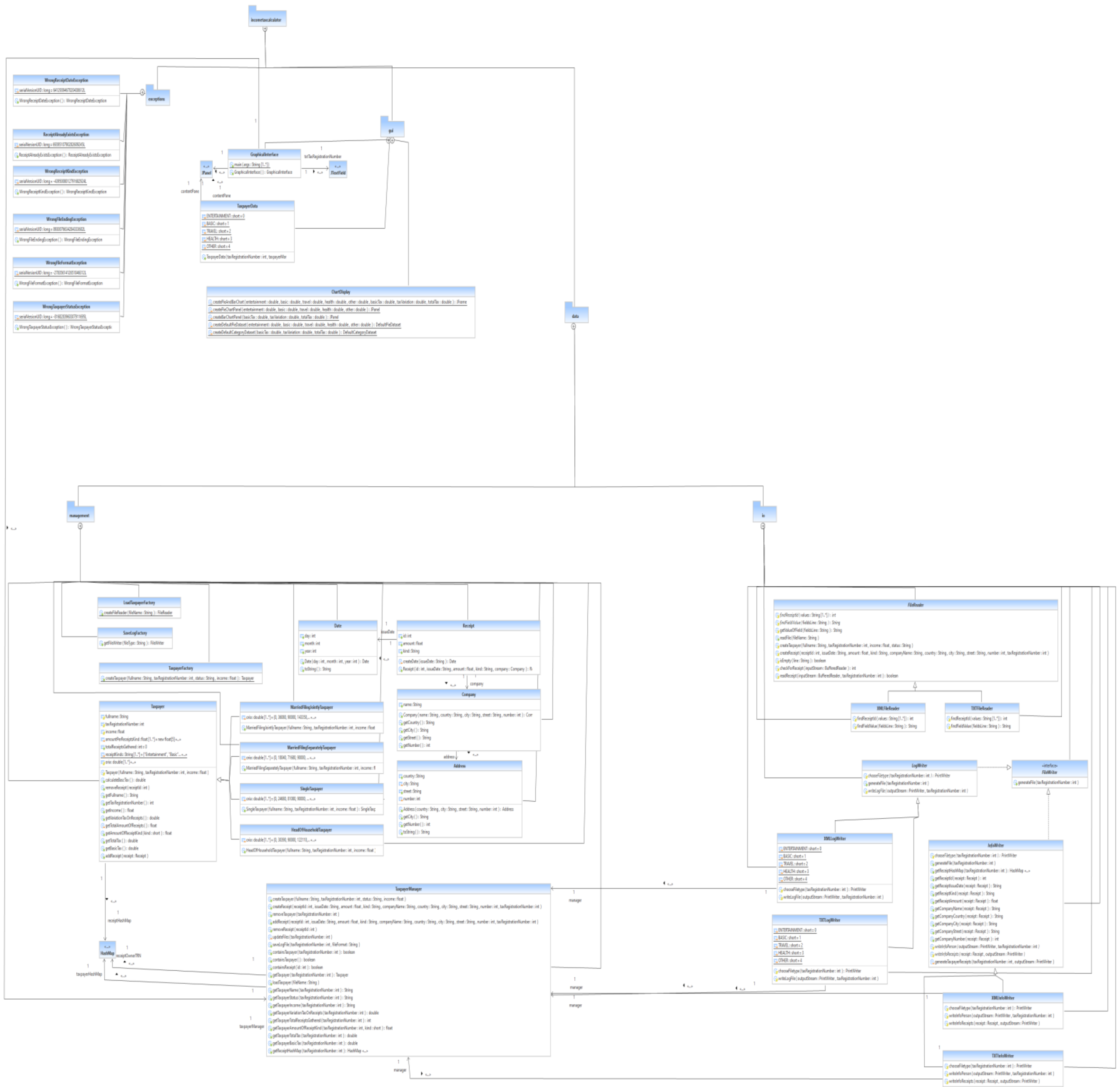
ARCHITECTURE

- The UML package diagram that shows the architecture of the **refactored** application.

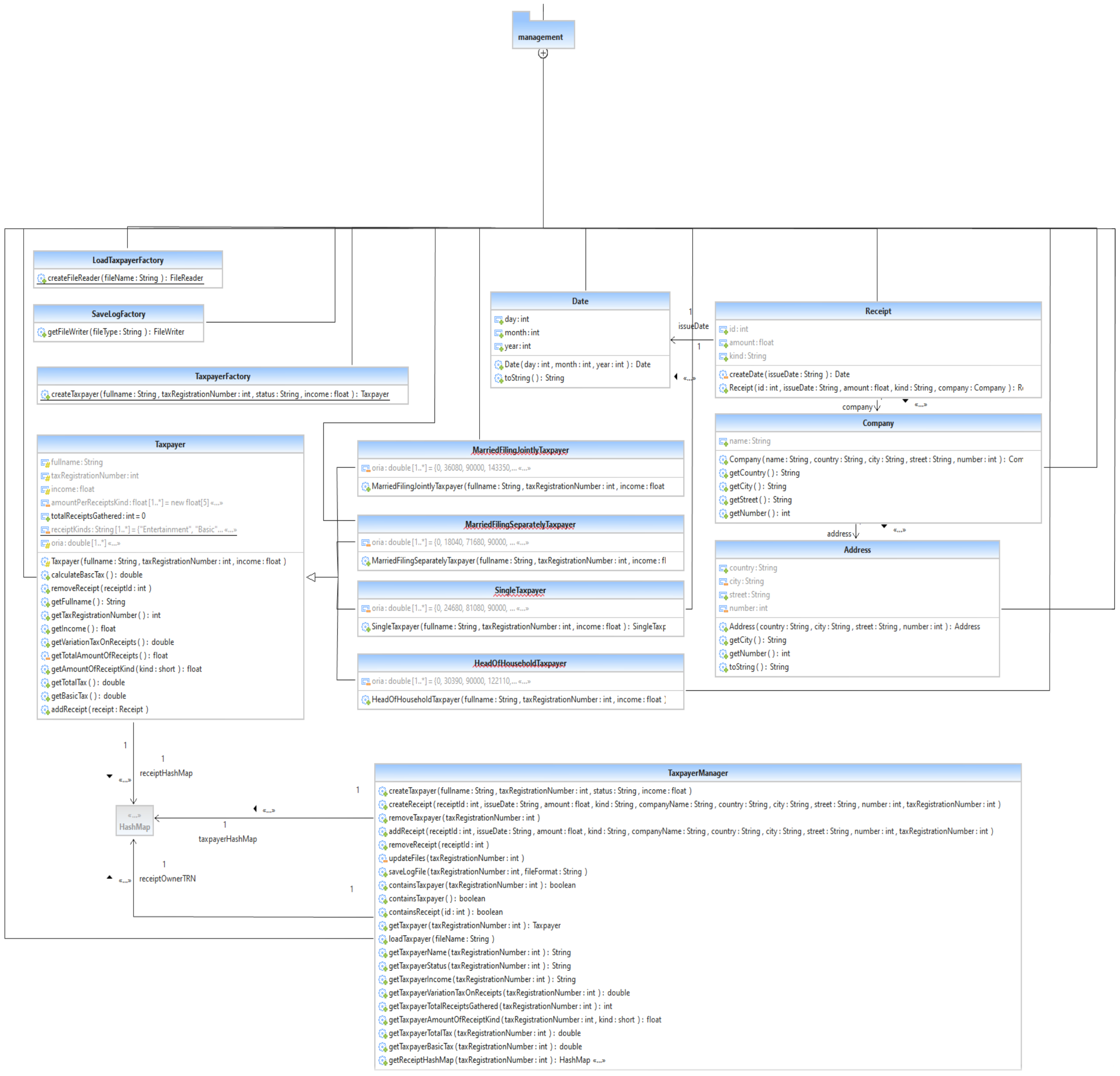


DETAILED DESIGN

- The UML class diagram of the application

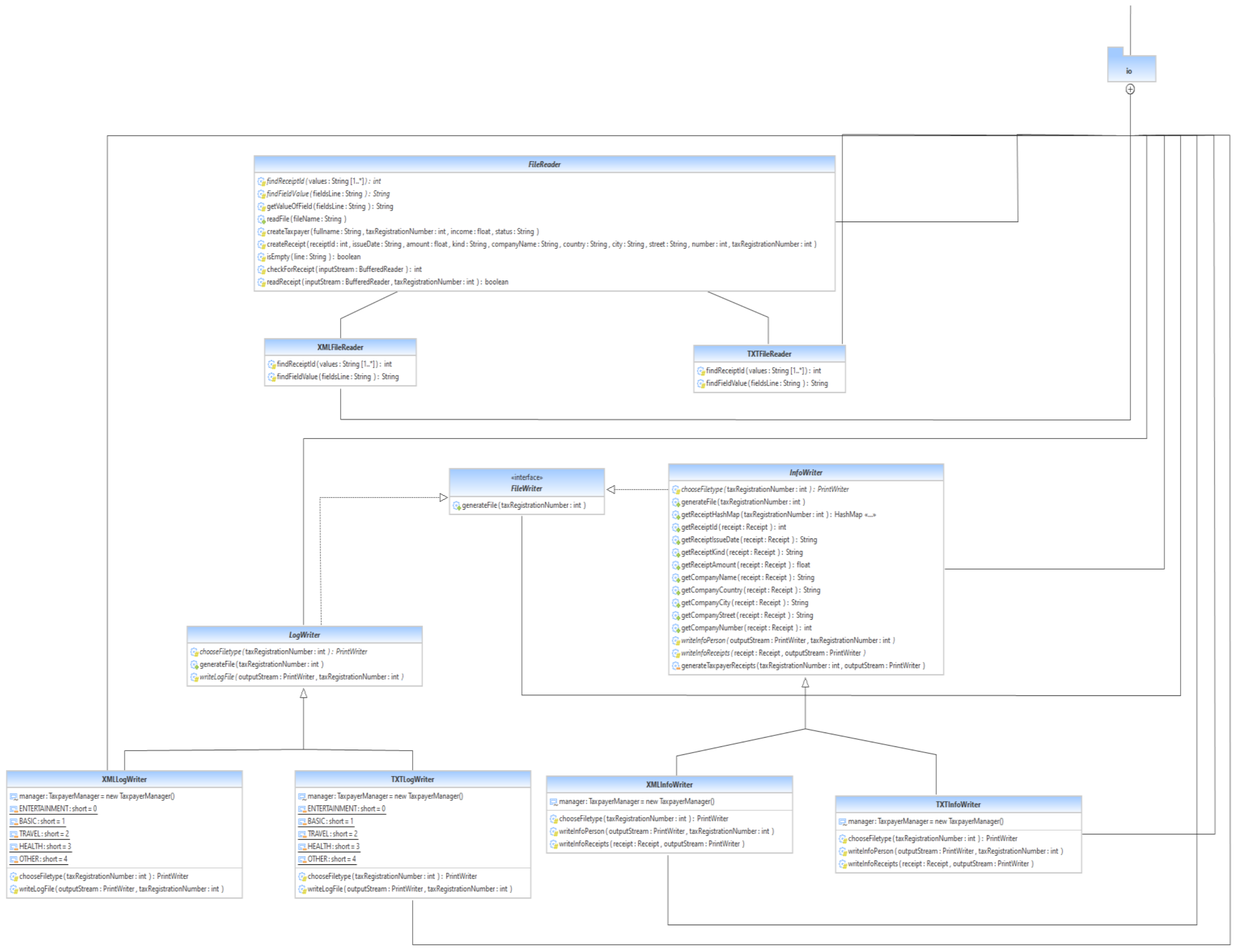


- The_UML class diagram for the **management** package of the application

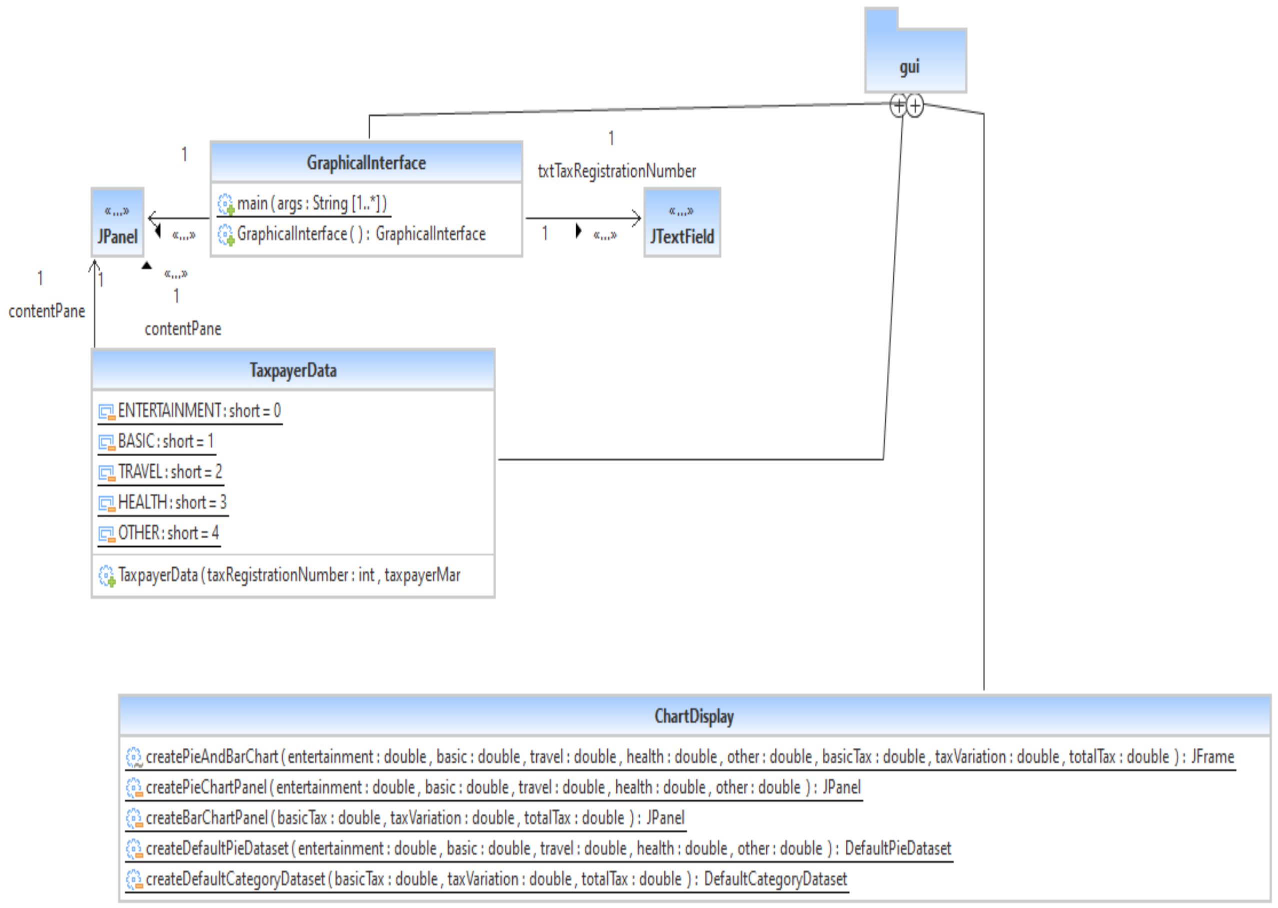


management
LoadTaxpayerFactory
createFileReader (fileName : String) : FileReader
SaveLogFactory
getFileWriter (fileType : String) : FileWriter
TaxpayerFactory
createTaxpayer (fullname : String , taxRegistrationNumber : int , status : String , income : float) : Taxpayer
Taxpayer
fullname : String
taxRegistrationNumber : int
income : float
amountPerReceiptsKind : float [1..*] = new float[5] «...»
totalReceiptsGathered : int = 0
receiptKinds : String [1..*] = {"Entertainment", "Basic"... «...»
oria : double [1..*] «...»
Taxpayer (fullname : String , taxRegistrationNumber : int , income : float)
calculateBascTax ( ) : double
removeReceipt (receiptId : int)
getFullname ( ) : String
getTaxRegistrationNumber ( ) : int
getIncome ( ) : float
getVariationTaxOnReceipts ( ) : double
getTotalAmountOfReceipts ( ) : float
getAmountOfReceiptKind (kind : short) : float
getTotalTax ( ) : double
getBasicTax ( ) : double
addReceipt (receipt : Receipt)
Date
day : int
month : int
year : int
Date (day : int , month : int , year : int) : Date
toString ( ) : String
1
issueDate
1
«...»
Receipt
id : int
amount : float
kind : String
createDate (issueDate : String) : Date
Receipt (id : int , issueDate : String , amount : float , kind : String , company : Company) : R
company
«...»
Company
name : String
Company (name : String , country : String , city : String , street : String , number : int) : Com
getCountry ( ) : String
getCity ( ) : String
getStreet ( ) : String
getNumber ( ) : int
address
«...»
Address
country : String
city : String
street : String
number : int
Address (country : String , city : String , street : String , number : int) : Address
getCity ( ) : String
getNumber ( ) : int
toString ( ) : String
MarriedFilingJointlyTaxpayer
oria : double [1..*] = {0, 36080, 90000, 143350,... «...»
MarriedFilingJointlyTaxpayer (fullname : String , taxRegistrationNumber : int , income : float
MarriedFilingSeparatelyTaxpayer
oria : double [1..*] = {0, 18040, 71680, 90000, ... «...»
MarriedFilingSeparatelyTaxpayer (fullname : String , taxRegistrationNumber : int , income : fl
SingleTaxpayer
oria : double [1..*] = {0, 24680, 81080, 90000, ... «...»
SingleTaxpayer (fullname : String , taxRegistrationNumber : int , income : float) : SingleTaxp
HeadOfHouseholdTaxpayer
oria : double [1..*] = {0, 30390, 90000, 122110,... «...»
HeadOfHouseholdTaxpayer (fullname : String , taxRegistrationNumber : int , income : float)
1
1
receiptHashMap
«...»
«...»
HashMap
«...»
1
taxpayerHashMap
1
receiptOwnerTRN
«...»
1
1
TaxpayerManager
createTaxpayer (fullname : String , taxRegistrationNumber : int , status : String , income : float)
createReceipt (receiptId : int , issueDate : String , amount : float , kind : String , companyName : String , country : String , city : String , street : String , number : int , taxRegistrationNumber : int)
removeTaxpayer (taxRegistrationNumber : int)
addReceipt (receiptId : int , issueDate : String , amount : float , kind : String , companyName : String , country : String , city : String , street : String , number : int , taxRegistrationNumber : int)
removeReceipt (receiptId : int)
updateFiles (taxRegistrationNumber : int)
saveLogFile (taxRegistrationNumber : int , fileFormat : String)
containsTaxpayer (taxRegistrationNumber : int) : boolean
containsTaxpayer ( ) : boolean
containsReceipt (id : int) : boolean
getTaxpayer (taxRegistrationNumber : int) : Taxpayer
loadTaxpayer (fileName : String)
getTaxpayerName (taxRegistrationNumber : int) : String
getTaxpayerStatus (taxRegistrationNumber : int) : String
getTaxpayerIncome (taxRegistrationNumber : int) : String
getTaxpayerVariationTaxOnReceipts (taxRegistrationNumber : int) : double
getTaxpayerTotalReceiptsGathered (taxRegistrationNumber : int) : int
getTaxpayerAmountOfReceiptKind (taxRegistrationNumber : int , kind : short) : float
getTaxpayerTotalTax (taxRegistrationNumber : int) : double
getTaxpayerBasicTax (taxRegistrationNumber : int) : double
getReceiptHashMap (taxRegistrationNumber : int) : HashMap «...»

- The UML class diagram for the **io** package of the application

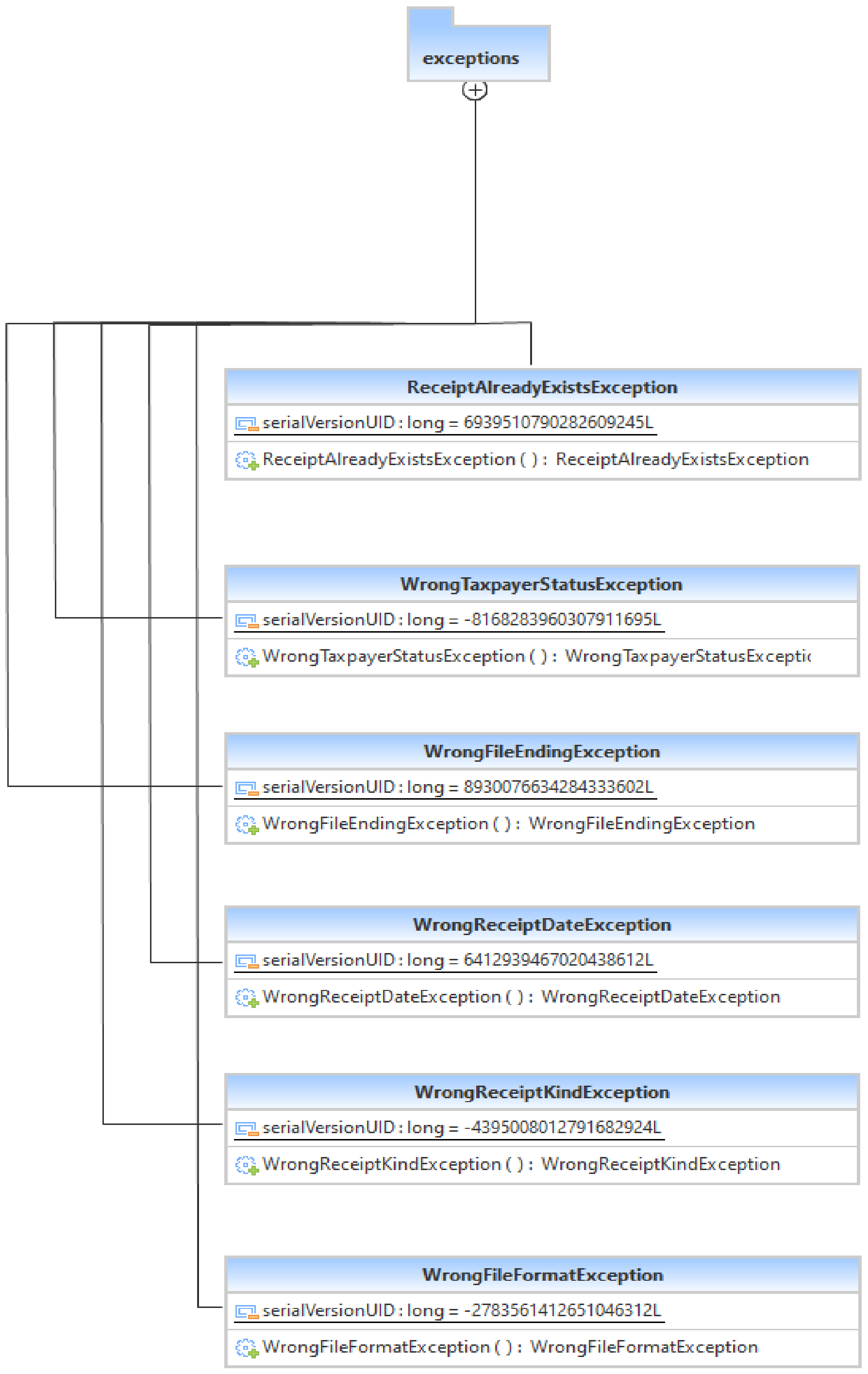

io
FileReader
findReceiptId (values : String [1..*]) : int
findFieldValue (fieldsLine : String) : String
getValueOfField (fieldsLine : String) : String
readFile (fileName : String)
createTaxpayer (fullname : String , taxRegistrationNumber : int , income : float , status : String)
createReceipt (receiptId : int , issueDate : String , amount : float , kind : String , companyName : String , country : String , city : String , street : String , number : int , taxRegistrationNumber : int)
isEmpty (line : String) : boolean
checkForReceipt (inputStream : BufferedReader) : int
readReceipt (inputStream : BufferedReader , taxRegistrationNumber : int) : boolean
XMLFileReader
findReceiptId (values : String [1..*]) : int
findFieldValue (fieldsLine : String) : String
TXTFileReader
findReceiptId (values : String [1..*]) : int
findFieldValue (fieldsLine : String) : String
«interface»
FileWriter
generateFile (taxRegistrationNumber : int)
InfoWriter
chooseFiletype (taxRegistrationNumber : int) : PrintWriter
generateFile (taxRegistrationNumber : int)
getReceiptHashMap (taxRegistrationNumber : int) : HashMap «...»
getReceiptId (receipt : Receipt) : int
getReceiptIssueDate (receipt : Receipt) : String
getReceiptKind (receipt : Receipt) : String
getReceiptAmount (receipt : Receipt) : float
getCompanyName (receipt : Receipt) : String
getCompanyCountry (receipt : Receipt) : String
getCompanyCity (receipt : Receipt) : String
getCompanyStreet (receipt : Receipt) : String
getCompanyNumber (receipt : Receipt) : int
writeInfoPerson (outputStream : PrintWriter , taxRegistrationNumber : int)
writeInfoReceipts (receipt : Receipt , outputStream : PrintWriter)
generateTaxpayerReceipts (taxRegistrationNumber : int , outputStream : PrintWriter)
LogWriter
chooseFiletype (taxRegistrationNumber : int) : PrintWriter
generateFile (taxRegistrationNumber : int)
writeLogFile (outputStream : PrintWriter , taxRegistrationNumber : int)
XMLLogWriter
manager : TaxpayerManager = new TaxpayerManager()
ENTERTAINMENT : short = 0
BASIC : short = 1
TRAVEL : short = 2
HEALTH : short = 3
OTHER : short = 4
chooseFiletype (taxRegistrationNumber : int) : PrintWriter
writeLogFile (outputStream : PrintWriter , taxRegistrationNumber : int)
TXTLogWriter
manager : TaxpayerManager = new TaxpayerManager()
ENTERTAINMENT : short = 0
BASIC : short = 1
TRAVEL : short = 2
HEALTH : short = 3
OTHER : short = 4
chooseFiletype (taxRegistrationNumber : int) : PrintWriter
writeLogFile (outputStream : PrintWriter , taxRegistrationNumber : int)
XMLInfoWriter
manager : TaxpayerManager = new TaxpayerManager()
chooseFiletype (taxRegistrationNumber : int) : PrintWriter
writeInfoPerson (outputStream : PrintWriter , taxRegistrationNumber : int)
writeInfoReceipts (receipt : Receipt , outputStream : PrintWriter)
TXTInfoWriter
manager : TaxpayerManager = new TaxpayerManager()
chooseFiletype (taxRegistrationNumber : int) : PrintWriter
writeInfoPerson (outputStream : PrintWriter , taxRegistrationNumber : int)
writeInfoReceipts (receipt : Receipt , outputStream : PrintWriter)

- The UML class diagram for the **gui** package of the application



gui
1
GraphicalInterface
main (args : String [1..*])
GraphicalInterface ( ) : GraphicalInterface
1
txtTaxRegistrationNumber
«...»
JPanel
«...»
1
«...»
«...»
JTextField
1
contentPane
1
«...»
1
contentPane
TaxpayerData
ENTERTAINMENT : short = 0
BASIC : short = 1
TRAVEL : short = 2
HEALTH : short = 3
OTHER : short = 4
TaxpayerData (taxRegistrationNumber : int , taxpayerMar
ChartDisplay
createPieAndBarChart (entertainment : double , basic : double , travel : double , health : double , other : double , basicTax : double , taxVariation : double , totalTax : double) : JFrame
createPieChartPanel (entertainment : double , basic : double , travel : double , health : double , other : double) : JPanel
createBarChartPanel (basicTax : double , taxVariation : double , totalTax : double) : JPanel
createDefaultPieDataset (entertainment : double , basic : double , travel : double , health : double , other : double) : DefaultPieDataset
createDefaultCategoryDataset (basicTax : double , taxVariation : double , totalTax : double) : DefaultCategoryDataset

- The UML class diagram for the **exceptions** package of the application


exceptions
ReceiptAlreadyExistsException
serialVersionUID : long = 6939510790282609245L
ReceiptAlreadyExistsException ( ) : ReceiptAlreadyExistsException
WrongTaxpayerStatusException
serialVersionUID : long = -8168283960307911695L
WrongTaxpayerStatusException ( ) : WrongTaxpayerStatusExceptic
WrongFileEndingException
serialVersionUID : long = 8930076634284333602L
WrongFileEndingException ( ) : WrongFileEndingException
WrongReceiptDateException
serialVersionUID : long = 6412939467020438612L
WrongReceiptDateException ( ) : WrongReceiptDateException
WrongReceiptKindException
serialVersionUID : long = -4395008012791682924L
WrongReceiptKindException ( ) : WrongReceiptKindException
WrongFileFormatException
serialVersionUID : long = -2783561412651046312L
WrongFileFormatException ( ) : WrongFileFormatException

REENGINEERING

Problem: Unnecessary complexity in Company class.

Solution: We examined the code in Company class and found that the method getAddress() is never used. We decided to remove it as it is unnecessary for the application.

Result: The Company class is now smaller and easier to understand.

Problem: Complex conditional logic in Taxpayer class.

Solution: We found that the methods addReceipt() and removeReceipt() use a lot of chained if-else statements to calculate totals and update the number of receipts. To simplify this, we created a private static final String array called receiptKinds in the Taxpayer class to hold the kinds of receipts. We replaced the if-else statements with a loop that iterates through the receiptKinds array, checking if the receipt kind matches any element of the array. We also modified the getVariationTaxOnReceipt() method in a similar way, by creating an array and looping through it.

Result: The Taxpayer class is now easier to read and understand as the contents of the comparisons are stored in simple arrays.

Problem: Duplicate code in the subclasses of the Taxpayer class.

Solution: We examined the code in the subclasses of Taxpayer class and found that they all use the same pattern in the calculateBasicTax() method, with the only difference being the income limits. To eliminate this duplication, we moved the method to the base Taxpayer class and created a private field for the limits. Lastly, we created constructors for each type of Taxpayer to initialize the limits with the correct values.

Result: The subclasses of the Taxpayer class no longer contain duplicate code. This will make the code easier to understand and maintain.

Problem:	Many responsibilities for the TaxpayerManager class.
Solution:	We found that the TaxpayerManager class has some methods that created different types of objects using if-else statements, making the code difficult to read. To solve this, we created some parameterized factories to handle the creation of different types of Taxpayer, FileWriter and FileReader objects.
Result:	The TaxpayerManger class is now smaller and more focused, with clearer responsibilities as we delegate some of its responsibilities to subordinate classes.
TO DO:	Missing factory for updateFiles() method.
Problem:	Duplicate code in TXTFileReader and XMLFileReader classes.
Solution:	We reviewed the code in the TXTFileReader and XMLFileReader classes and found that both construct two methods (checkForReceipt() and getValueOfField()) that share similar code. To solve this issue, we moved the shared code to the base class FileReader and created two abstract methods (findReceiptId() and findFieldValue()) that should be implemented in the subclasses and contain the unique code for the checkForReceipt() and getValueOfField() methods.
Result:	This refactoring made the TXTFileReader andXMLFileReader classes smaller and easier to understand, eliminated duplication of code, and made the design more maintainable and efficient.
Problem:	Refuse Bequest in FileWriter class.
Solution:	We reviewed the code in the FileWriter class and found that there were several methods that were only used by some the subclasses. To fix this problem, we moved these methods to the subclasses (TXTInfoWriter, XMLInfoWriter, TXTLogWriter and XMLLogWriter) that need them and also moved the methods that were used directly by the TaxpayerManager to that class. Also, we changed the FileWriter class to an interface.
Result:	This improved the design and increased the cohesion and coupling of the classes.

Problem: Duplicate code in TXTInfoWriter and XMLInfoWriter classes.

Solution: We examined the code in TXTInfoWriter and XMLInfoWriter classes and found that after refactoring the FileWriter class we had created similar methods in its previous subclasses. We also noticed that their core methods share similar code. So, we decided to create an abstract InfoWriter super class that implements the FileWriter interface and holds the duplicate methods. Then, we moved the shared code from the core methods in the TXTInfoWriter and XMLInfoWriter classes to the FileWriter superclass and created abstract methods that should be implemented in the subclasses and hold the unique code for each core method.

Result: This allowed us to eliminate the duplication of code and make the design more maintainable and efficient.

Problem: Duplicate code in TXTLogWriter and XMLLogWriter classes.

Solution: Similarly, to the TXTInfoWriter and XMLInfoWriter classes, the TXTLogWriter and XMLLogWriter classes also had duplicate code, with the core algorithms of their methods being similar but only differing in the constant string tags that were written along with the basic information. To fix this problem we created template methods in an abstract LogWriter super class that implements the FileWriter interface and abstracted the parts of the code that were different by creating simple abstract methods that were implemented in the subclass.

Result: This eliminated duplication of code and made the design more maintainable.

Problem: Unnecessarily difficult browsing of the application.

Solution: From the very first time that we run the application we found unnecessarily difficult to use the application's basic functions. To fix this we changed the way that the taxpayers are selected to view their details or to delete their data. From entering the number every time to be able to select their number from the list.in the same way we change the delete operation of a receipt .Also we integrate the results in one window and change the color scheme of the application to be less tiring for the eyes .

Result: Those changes simplifying the browsing of the application by making it easier , faster and increases the productivity of the tax accountan

CLASSES RESPONSIBILITIES AND COLLABORATIONS (CRC CARDS)

Class Name: FileReader	
Responsibilities	**Collaborations**
- Is responsible for reading the loaded file. - Creates the loaded Taxpayer and his receipts.	- Depends on the WrongTaxpayerStatusException, WrongFileFormatException, WrongReceiptKindException and WrongReceiptDateException classes to throw an appropriate error message. - Depends on the TaxpayerManager class to create a Taxpayer object and his receipts. - Is necessary for the TXTFileReader and XMLFileReader classes because they extend it.

Class Name: FileWriter	
Responsibilities	**Collaborations**
- Provides a method that should be overridden by the classes that implements this class.	- Is necessary for the InfoWriter and LogWriter classes because they implement it.

Class Name: InfoWriter	
Responsibilities	**Collaborations**
- Provides some methods that should be overridden by the classes that extend this class. - Generates a file containing information about the taxpayer and his receipts.	- Depends on the FileWriter interface because it implements it. - Depends on the Receipt class to create Receipt objects or use them as parameters for its methods. - Depends on the TaxpayerManager class to create a Taxpayer object to get his receipts.

	- Is necessary for the TXTInfoWriter and XMLInfoWriter classes because they extend it.

Class Name: LogWriter	
Responsibilities	**Collaborations**
- Provides some methods that should be overridden by the classes that extend this class. - Generates a log file containing information about the taxpayer and his receipts.	- Depends on the FileWriter interface because it implements it. - Is necessary for the TXTLogWriter and XMLLogWriter classes because they extend it.

Class Name: TXTFileReader	
Responsibilities	**Collaborations**
- Finds the receipt's ID given an array of Strings. - Finds the value of a field given a String.	- Depends on the FileReader class because it extends it. - Depends on the WrongFileFormat class to throw an appropriate error message. - Is necessary for the FileReader class because it returns the receipt's id and values for a field for it.

Class Name: TXTInfoWriter	
Responsibilities	**Collaborations**
- Writes information about a Taxpayer to a txt file. - Writes information about a Taxpayer's receipts to a txt file.	- Depends on the InfoWriter class because it extends it. - Depends on the TaxpayerManager class to get a Taxpayer's information. - Depends on the Receipt class to get a Receipt's information.

	- Is necessary for the InfoWriter class because its methods write strings for the generated info file.

Class Name: TXTLogWriter	
Responsibilities	**Collaborations**
- Writes information about a Taxpayer, his receipts' kinds and his tax to a txt file.	- Depends on the LogWriter class because it extends it. - Depends on the TaxpayerManager class to get a Taxpayer's information. - Is necessary for the LogWriter class because its method write strings for the generated log file.

Class Name: XMLFileReader	
Responsibilities	**Collaborations**
- Finds the receipt's ID given an array of Strings. - Finds the value of a field given a String.	- Depends on the FileReader class because it extends it. - Depends on the WrongFileFormat class to throw an appropriate error message. - Is necessary for the FileReader class because it returns the receipt's id and values for a field for it.

Class Name: XMLInfoWriter	
Responsibilities	**Collaborations**
- Writes information about a Taxpayer to a xml file. - Writes information about a Taxpayer's receipts to a xml file.	- Depends on the InfoWriter class because it extends it. - Depends on the TaxpayerManager class to get a Taxpayer's information.

	- Depends on the Receipt class to get a Receipt's information. - Is necessary for the InfoWriter class because its methods write strings for the generated info file.

Class Name: XMLLogWriter	
Responsibilities	**Collaborations**
- Writes information about a Taxpayer, his receipts' kinds and his tax to a xml file.	- Depends on the LogWriter class because it extends it. - Depends on the TaxpayerManager class to get a Taxpayer's information. - Is necessary for the LogWriter class because its method write strings for the generated log file.

Class Name: Address	
Responsibilities	**Collaborations**
- Stores information about a company's address	- Is necessary for the Company class because it stores information about a company's address.

Class Name: Company	
Responsibilities	**Collaborations**
- Stores information about a company's name and address.	- Depends on the Address class to store the company's address. - Is necessary for the TaxpayerManager class because a company is needed for creating a Receipt.

Class Name: Date	
Responsibilities	**Collaborations**
- Stores information about a receipt’s issue date.	- Is necessary for the Receipt class because a receipt needs an issue date.

Class Name: HeadOfHouseholdTaxpayer	
Responsibilities	**Collaborations**
- Creates a HeadOfHouseholdTaxpayer object (kind of Taxpayer).	- Depends on the Taxpayer class because it extends it. - Is necessary for the TaxpayerFactory class because it creates a specific Taxpayer object based on his status.

Class Name: LoadTaxpayerFactory	
Responsibilities	**Collaborations**
- Creates a specific FileReader object based on a loaded filename’s ending.	- Depends on the FileReader class because it creates a specific FileReader object. - Depends on the TXTFileReader and XMLFileReader classes because it creates a TXTFileReader or a XMLFileReader object. - Depends on the WrongFileEndingException class to throw an appropriate error message. - Is necessary for the TaxpayerManager class because it returns a FileReader object needed for loading a Taxpayer file.

Class Name: MarriedFilingJointlyTaxpayer	
Responsibilities	**Collaborations**
- Creates a MarriedFilingJointlyTaxpayer object (kind of Taxpayer).	- Depends on the Taxpayer class because it extends it. - Is necessary for the TaxpayerFactory class because it creates a specific Taxpayer object based on his status.

Class Name: MarriedFilingSeparatelyTaxpayer	
Responsibilities	**Collaborations**
- Creates a MarriedFilingSeparatelyTaxpayer object (kind of Taxpayer).	- Depends on the Taxpayer class because it extends it. - Is necessary for the TaxpayerFactory class because it creates a specific Taxpayer object based on his status.

Class Name: Receipt	
Responsibilities	**Collaborations**
- Stores information about a Receipt.	- Depends on the Company class because a receipt holds information about the company that issued it. - Depends on the Date class because a receipt holds information about what date was it issued. - Depends on the WrongReceiptDateException class to throw an appropriate error message. - Is necessary for the TaxpayerManager class because is needed for creating a receipt or removing a Taxpayer and his receipts. - Is necessary for the Taxpayer class because is needed for adding or removing a receipt. - Is necessary for the InfoWriter class because is needed for

	finding a receipt’s information in order to be written in an info file. - Is necessary for the TaxpayerData class to view a Taxpayer’s receipts.

Class Name: SaveLogFactory	
Responsibilities	**Collaborations**
- Creates a specific FileWriter object in order to write to a generated log file.	- Depends on the FileWriter class because it creates a specific FileWriter object. - Depends on the TXTLogWriter and XMLLogWriter classes because it creates a TXTLogWriter or a XMLLogWriter object. - Depends on the WrongFileFormatException class to throw an appropriate error message. - Is necessary for the TaxpayerManager class in order to write to a generated log file.

Class Name: SingleTaxpayer	
Responsibilities	**Collaborations**
- Creates a SingleTaxpayer object (kind of Taxpayer).	- Depends on the Taxpayer class because it extends it. - Is necessary for the TaxpayerFactory class because it creates a specific Taxpayer object based on his status.

Class Name: Taxpayer	
Responsibilities	**Collaborations**

- Stores information about a Taxpayer. - Calculates a Taxpayer’s basic tax. - Calculates a Taxpayer’s increase or decrease of tax based on his receipts. - Calculates a Taxpayer’s total amount of receipts. - Calculates a Taxpayer’s final tax.	- Depends on the WrongReceiptKindException class to throw an appropriate error message. - Depends on the Receipt class to calculate a Taxpayer’s total receipts and amount per kind. - Is necessary for the MarriedFilingSeparatelyTaxpayer, MarriedFilingJointlyTaxpayer, HeadOfHouseholdTaxpayer and SingleTaxpayer classes because they extend it. - Is necessary for the TaxpayerManager class - to create or remove a receipt - to get a Taxpayer’s information

Class Name: TaxpayerFactory	
Responsibilities	**Collaborations**
- Creates a Taxpayer object based on his status.	- Depends on the WrongTaxpayerStatusException class to throw an appropriate error message. - Depends on the Taxpayer class to create a specific Taxpayer object based on his status. - Depends on the MarriedFilingSeparatelyTaxpayer, MarriedFilingJointlyTaxpayer, HeadOfHouseholdTaxpayer and SingleTaxpayer classes because it creates a Taxpayer object.

Class Name: TaxpayerManager	
Responsibilities	**Collaborations**
- Creates Taxpayer objects. - Removes a Taxpayer based on his AFM.	- Depends on the WrongTaxpayerStatusException, WrongReceiptKindException, WrongFileFormatException, ReceiptAlreadyExistsException, WrongFileEndingException and WrongReceiptDateException classes to throw an

- Adds and creates receipts for a Taxpayer. - Removes receipt based on its id. - Updates files when changes happen. - Saves a log file with info about a Taxpayer and his receipts. - Checks if a taxpayer exists. - Checks if a receipt exists. - Loads a taxpayer from a loaded file. - Finds a taxpayer's status. - Finds a taxpayer's income, tax and receipts.	appropriate error message. - Depends on the TaxpayerFactory class to create a specific Taxpayer object based on his status. - Depends on the Receipt class to create, add or remove receipts. - Depends on the Taxpayer class - to create or remove a Taxpayer - to find information about a Taxpayer - Depends on the SaveLogFactory to create a FileWriter object to write to a generated log file. - Depends on the TXTInfoWriter and XMLInfoWriter classes to write the updated information in the generated info file. - Depends on the FileReader class to create a FileReader object to load a Taxpayer info file. - Depends on the MarriedFilingSeparatelyTaxpayer, MarriedFilingJointlyTaxpayer, HeadOfHouseholdTaxpayer and SingleTaxpayer classes to find a Taxpayers status. - Is necessary for the FileReader class to create a Taxpayer or a receipt for a Taxpayer. - Is necessary for the InfoWriter class to find the receipts of a Taxpayer. - Is necessary for the TXTInfoWriter and XMLInfoWriter classes to write information about the taxpayer and his receipt to the info file. - Is necessary for the TXTLogWriter and XMLLogWriter classes to write information about the taxpayer and his taxes/receipts kind in the generated log file. - Is necessary for the GraphicalInterface class to load, find or remove a Taxpayer. - Is necessary for the TaxpayerData class

	- to add or remove or get (a) receipt(s) for a Taxpayer by the user of the application. - to save a log file about the Taxpayer and his taxes/ receipts/ information by the user of the application. - to get and view information about the Taxpayer - to view reports about the Taxpayer's info and taxes

Class Name: ReceiptAlreadyExistsException	
Responsibilities	**Collaborations**
- Provides an error message when a receipt already exists.	- Depends on the Exception class because it extends it. - Is necessary for the TaxpayerManager when it creates new receipts.

Class Name: WrongFileEndingException	
Responsibilities	**Collaborations**
- Provides an error message when the loaded file is not txt or xml.	- Depends on the Exception class because it extends it. - Is necessary for the LoadTaxpayerFactory class when it creates new FileReader objects.

Class Name: WrongFileFormatException	
Responsibilities	**Collaborations**
- Provides an error message when generated log file is not a txt or xml type. - Provides an error message when reading the loaded	- Depends on the Exception class because it extends it. - Is necessary for the SaveLogFactory class when the

file and find wrong formatting.	generated log file is neither txt nor xml. - Is necessary for the FileReader class when there is invalid formatting in the loaded info file.

Class Name: WrongReceiptDateException	
Responsibilities	**Collaborations**
- Provides an error message when the receipt's issue date is wrong formatted.	- Depends on the Exception class because it extends it. - Is necessary for the Receipt class when it creates new receipts.

Class Name: WrongReceiptKindException	
Responsibilities	**Collaborations**
- Provides an error message when the receipt's given kind doesn't exist. - Provides an error message when the user tries to remove a receipt that has invalid kind type.	- Depends on the Exception class because it extends it. - Is necessary for the Taxpayer class when it creates a receipt with non-exist kind or when the user tries to remove a receipt that has invalid kind type.

Class Name: WrongTaxpayerStatusException	
Responsibilities	**Collaborations**
- Provides an error message when a Taxpayer is created with invalid status.	- Depends on the Exception class because it extends it. - Is necessary for the TaxpayerFactory class when it creates a taxpayer with a non-existed status.

Class Name: ChartDisplay	
Responsibilities	**Collaborations**
- Create a pie chart that displays the percentage of the total amount of each kind of receipt. - Create a bar chart that displays tax analysis data of the Taxpayer.	- Is necessary for the TaxpayerData class for displaying these charts in the GUI when the user press the button View Report.

Class Name: GraphicalInterface	
Responsibilities	**Collaborations**
- Starts the application. - Allows the user to load a file that has information about a taxpayer and his receipts via typing his AFM. - Allows the user to choose the type of the file (txt or xml). - Allows the user to select a taxpayer from a list via his AFM and display his information and receipts. - Allows the user to select a taxpayer from a list via his AFM and delete him. - Displays appropriate error messages when - the user tries to load an invalid / non – existing file. - the user tries to load an existing Taxpayer. - the user tries to display or remove a Taxpayer without selecting a Taxpayer’s AFM from the list first. - the user tries to display or remove a Taxpayer while the list of Taxpayers is empty.	- Depends on the WrongTaxpayerStatusException, WrongReceiptKindException, WrongFileFormatException, WrongFileEndingException and WrongReceiptDateException classes to throw an appropriate error message. - Depends on the TaxpayerManager class to load, find or remove a Taxpayer.

Class Name: TaxpayerData	
Responsibilities	**Collaborations**
- Displays the selected Taxpayer's information and receipts. - Allows the user to add a receipt for a Taxpayer. - Allows the user to select and delete a receipt for the selected Taxpayer. - Allows the user to view a pie chart that displays the percentage of the total amount of each kind of receipt and a bar chart that displays tax analysis data of the Taxpayer. - Allows the user to save into a log file the Taxpayer's information. - Displays appropriate error messages when - the user tries to add a receipt with invalid information format. - the user tries to add an existing receipt. - the user tries to remove a receipt without selecting one from the list first. - the user tries to remove a receipt while the list of receipts is empty.	- Depends on the WrongReceiptKindException, WrongFileFormatException, ReceiptAlreadyExistsException and WrongReceiptDateException classes to throw an appropriate error message. - Depends on the ChartDisplay class for displaying the charts in the GUI when the user press the View Report button. - Depends on the TaxpayerManager class - to find a Taxpayer. - to add find, add and remove receipts for the Taxpayer. - to find Taxpayer's amount of receipt kinds and tax and display them via the ChartDisplay method. - Depends on the Receipt class to find the Taxpayer's receipts. - Is necessary for the GraphicalInterface class for displaying the selected Taxpayer's information and actions.