MINNESOTA INCOME TAX CALCULATION

PROJECT

PHASE 2 - REENGINEERING THE CODE

ΠΕΡΙΚΛΗΣ ΙΩΑΝΝΟΥ , 2880

ΙΩΑΝΝΗΣ ΒΑΣΙΛΕΙΟΥ ,2647

GOAL OF THE PROJECT

The goal of this project is to re-engineer a Java application. At a glance, the application serves for **the income tax calculation of the Minnesota state citizens**. The tax calculation accounts for the marital status of a given citizen, his income, and the amount of money that he has spend, as witnessed by a set of receipts declared along with the income. The legacy application takes as **input txt** or **xml** files that contain the necessary data for each citizen. The tax calculation is based on a complex algorithm provided by the Minnesota state. The application further produces graphical representations of the data in terms of **bar and pie charts**. Finally. the application produces respective **output** reports in **txt** or **xml.**

TABLE OF CONTENTS

INTRODUCTION

✔ **The role/responsibilities of each class :**

- **Testing** : we prepare test cases that will allow you to understand and test the inner workings of the classes. Use JUnit for the tests. More specifically, develop JUnit tests that automatically test the back end of the application (i.e. the incometaxcalculator.data.io and the incometaxcalculator.data.management classes). Test the following functionalities :

  1. Tax calculation algorithms
  2. Input facilities
  3. Output facilities

  We specify the application architecture in terms of a **UML** package diagram. Specify the detailed design in terms of UML class diagrams. Prepare **CRC** cards that describe the responsibilities and collaborations of each class.

✔ **Eclipse CheckStyle** plugin and looking for more general problems as :

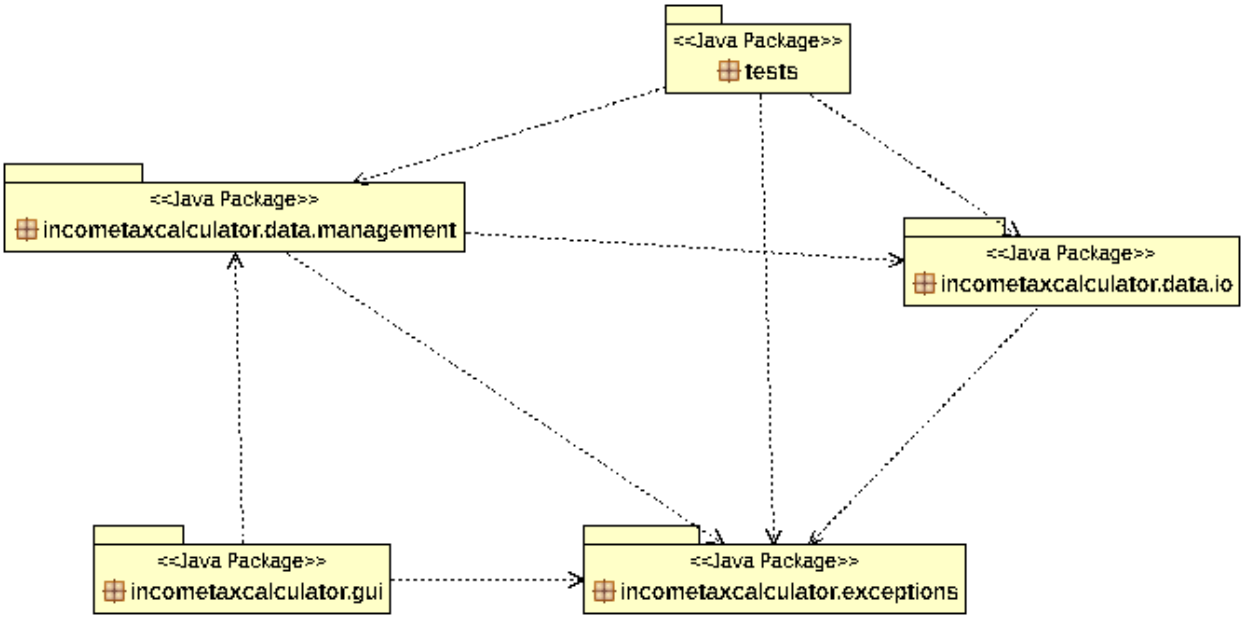
1. Detect possibilities of problematic classes with many responsibilities.
2. Detect possibilities of problematic classes with very few responsibilities.
3. Detect possibilities of similar classes/methods with duplicated code.

NOTE (1) : The option in order to fix our style and size problems , **Source>Format facility** does not work for us **.**

NOTE (2) : There are some checkstyle problems , as more than 2 parameters in some methods , more than the max number of lines in some methods or the size of some classes. Because of our limited time we were not able to fix them.**BUT** we suppose we will send to you the improved project if we read all the lessons we expect at **CHRISTMAS HOLIDAYS**.

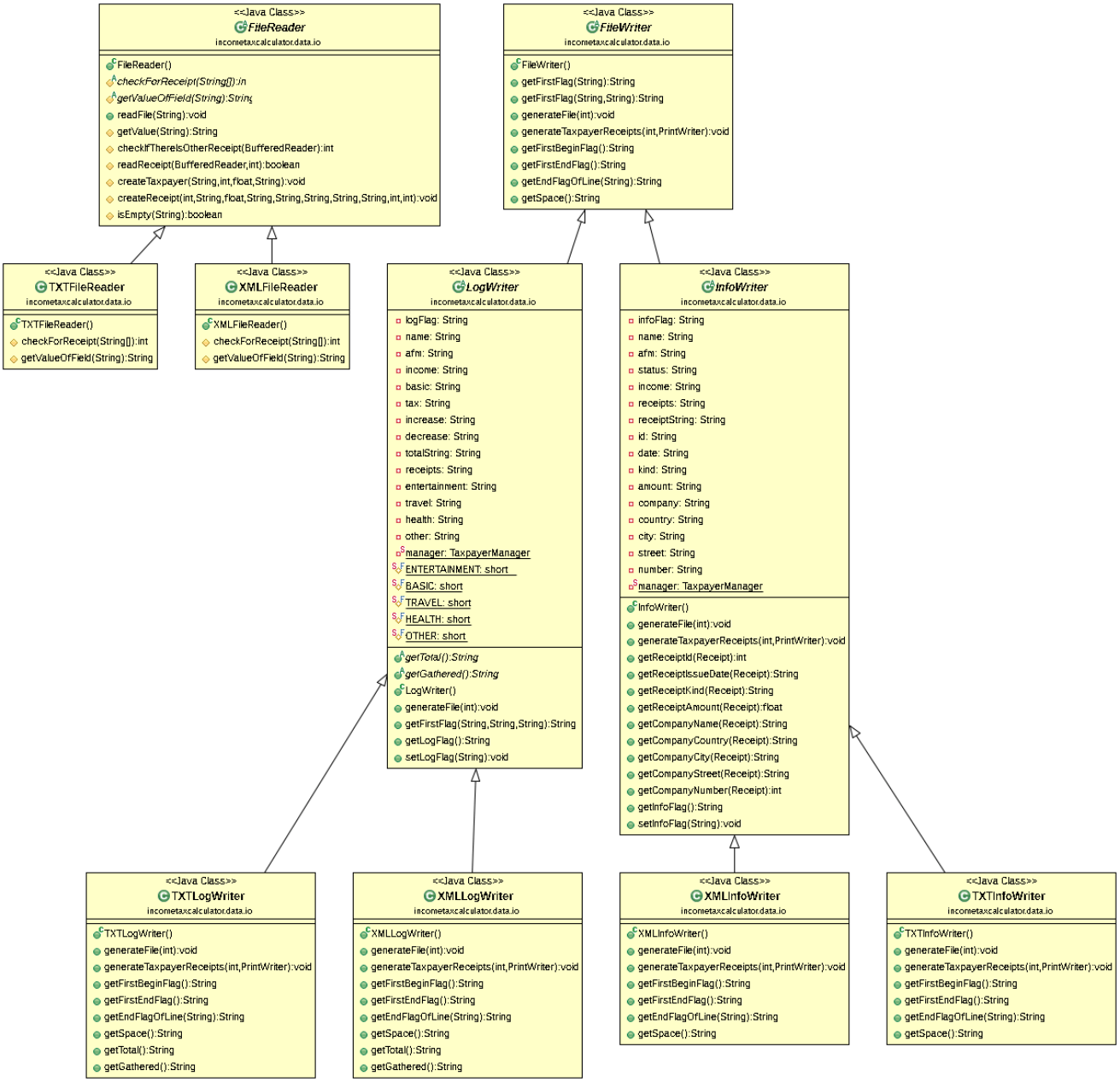
DESIGN RECOVERY

ARCHITECTURE

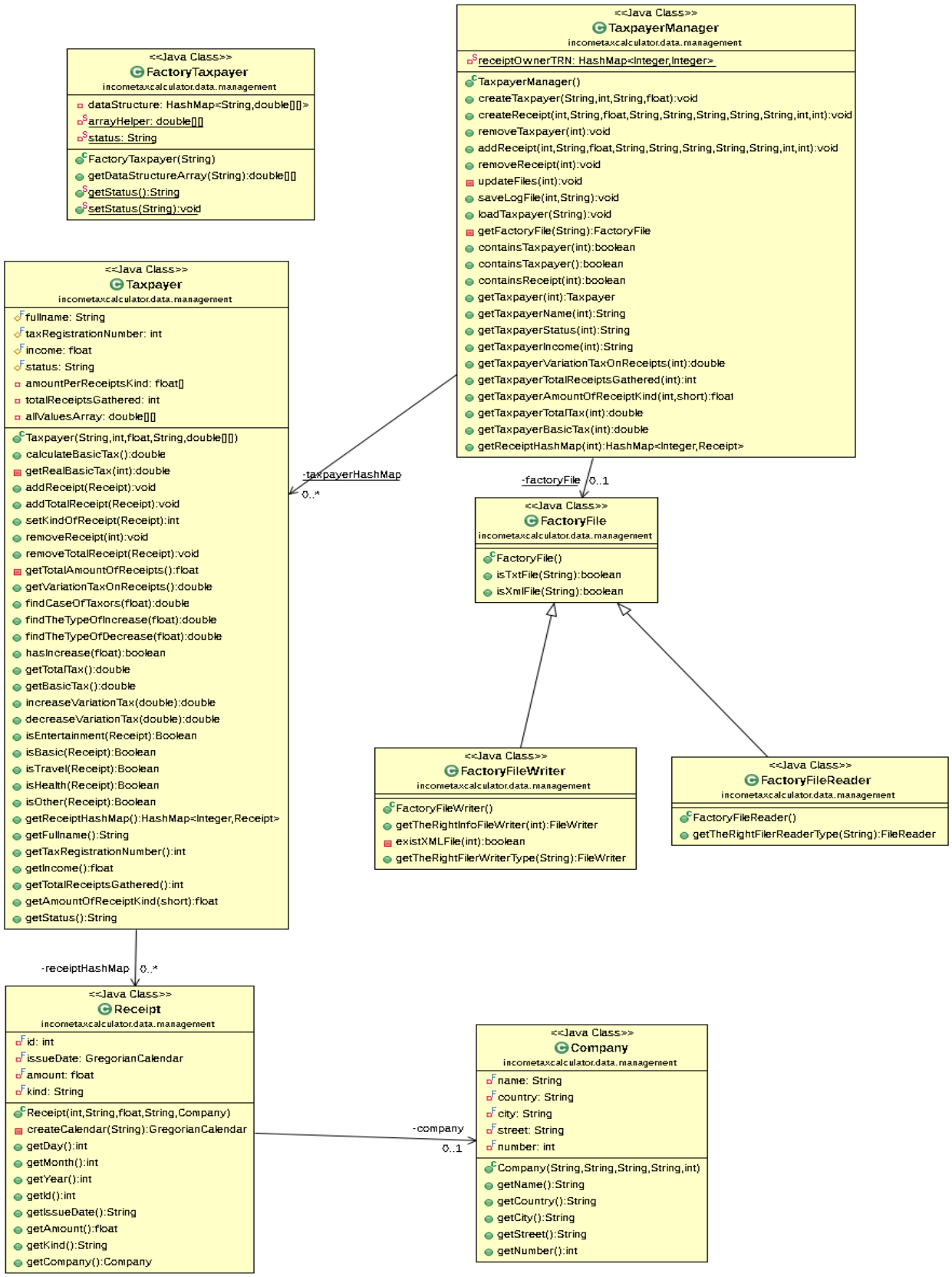


DETAILED DESIGN

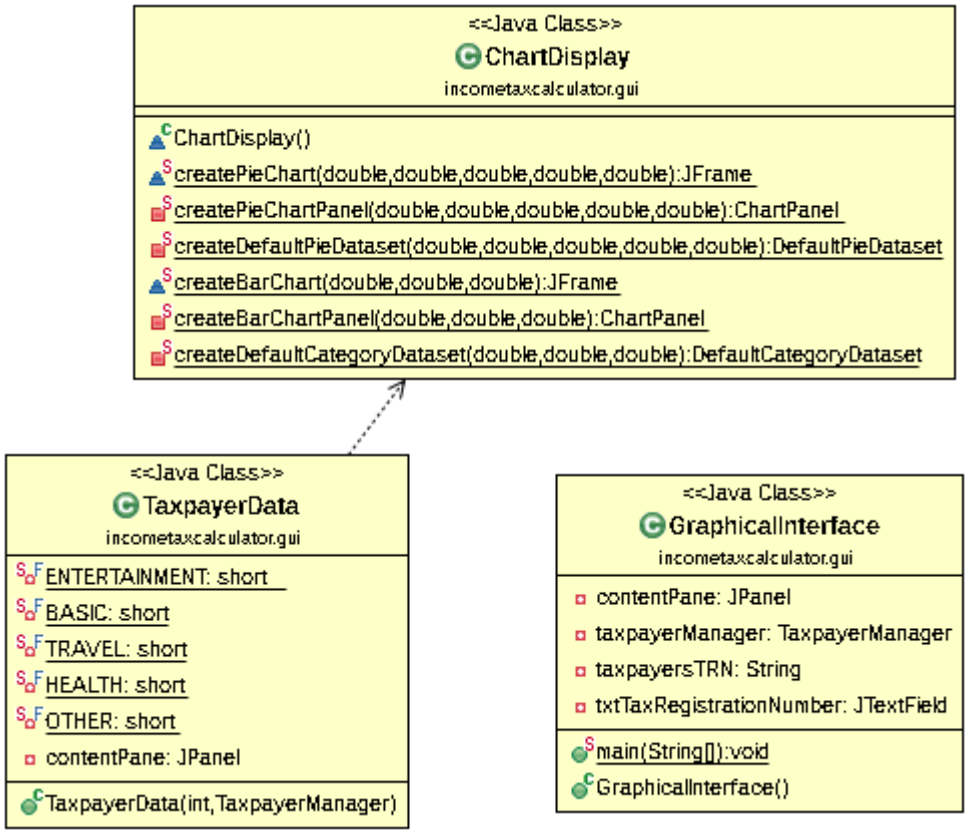
- IncomeTaxCalculatorDataIo - Uml :



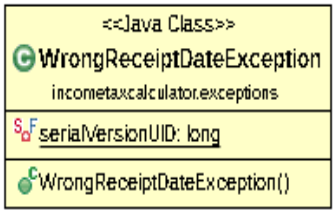
- IncomeTaxCalculatorDataManagement - Uml :

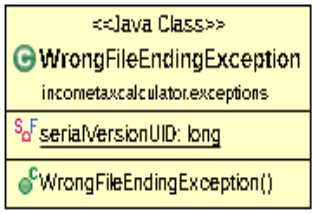

<<Java Class>>
FactoryTaxpayer
incometaxcalculator.data.management
dataStructure: HashMap<String,double[][]>
arrayHelper: double[][]
status: String
FactoryTaxpayer(String)
getDataStructureArray(String):double[][]
getStatus():String
setStatus(String):void
<<Java Class>>
TaxpayerManager
incometaxcalculator.data.management
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
loadTaxpayer(String):void
getFactoryFile(String):FactoryFile
containsTaxpayer(int):boolean
containsTaxpayer():boolean
containsReceipt(int):boolean
getTaxpayer(int):Taxpayer
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
status: String
amountPerReceiptsKind: float[]
totalReceiptsGathered: int
allValuesArray: double[][]
Taxpayer(String,int,float,String,double[][])
calculateBasicTax():double
getRealBasicTax(int):double
addReceipt(Receipt):void
addTotalReceipt(Receipt):void
setKindOfReceipt(Receipt):int
removeReceipt(int):void
removeTotalReceipt(Receipt):void
getTotalAmountOfReceipts():float
getVariationTaxOnReceipts():double
findCaseOfTaxors(float):double
findTheTypeOfIncrease(float):double
findTheTypeOfDecrease(float):double
hasIncrease(float):boolean
getTotalTax():double
getBasicTax():double
increaseVariationTax(double):double
decreaseVariationTax(double):double
isEntertainment(Receipt):Boolean
isBasic(Receipt):Boolean
isTravel(Receipt):Boolean
isHealth(Receipt):Boolean
isOther(Receipt):Boolean
getReceiptHashMap():HashMap<Integer,Receipt>
getFullname():String
getTaxRegistrationNumber():int
getIncome():float
getTotalReceiptsGathered():int
getAmountOfReceiptKind(short):float
getStatus():String
-taxpayerHashMap
0..*
-factoryFile
0..1
<<Java Class>>
FactoryFile
incometaxcalculator.data.management
FactoryFile()
isTxtFile(String):boolean
isXmlFile(String):boolean
<<Java Class>>
FactoryFileWriter
incometaxcalculator.data.management
FactoryFileWriter()
getTheRightInfoFileWriter(int):FileWriter
existXMLFile(int):boolean
getTheRightFilerWriterType(String):FileWriter
<<Java Class>>
FactoryFileReader
incometaxcalculator.data.management
FactoryFileReader()
getTheRightFilerReaderType(String):FileReader
-receiptHashMap
0..*
<<Java Class>>
Receipt
incometaxcalculator.data.management
id: int
issueDate: GregorianCalendar
amount: float
kind: String
Receipt(int,String,float,String,Company)
createCalendar(String):GregorianCalendar
getDay():int
getMonth():int
getYear():int
getId():int
getIssueDate():String
getAmount():float
getKind():String
getCompany():Company
-company
0..1
<<Java Class>>
Company
incometaxcalculator.data.management
name: String
country: String
city: String
street: String
number: int
Company(String,String,String,String,int)
getName():String
getCountry():String
getCity():String
getStreet():String
getNumber():int

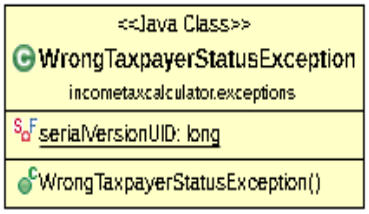
- IncomeTaxCalcilatorGui – Uml :

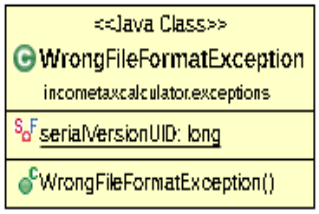


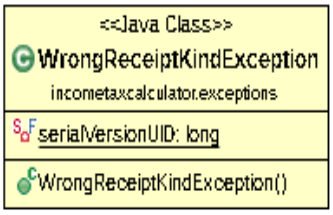
- IncomeTaxCalcilatorExceptions – Uml :

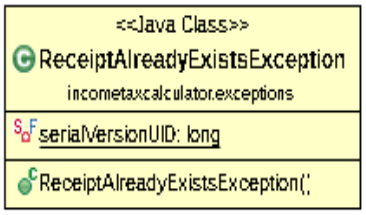












IMPLEMENTATION

- **<u>Package :</u> incometaxcalculator.data.management**

Class Name: TaxpayerManager	
Responsibilities	**Collaborations**
• Pepresents the functions that the classes inherited from them can implement in their own way. • Holds a hashmap (receiptOwnerTRN) all the taxpayers and their proofs • Holds a taxpayer (taxpayer HashMap) to the taxpayers and their afm	• Taxpayer • Receipt • Company • FactoryFile • FactoryTaxpayer

Class Name: Taxpayer	
Responsibilities	**Collaborations**

• Keeps a hashMap (receipt HashMap) the proof Id and the corresponding receipt item • Has methods used to process evidence • Defines the calculateBasicTax () method • Has some information from the receipts	• TaxpayerManager • Receipt • Company

Class Name: Receipt	
Responsibilities	**Collaborations**
• Keeps the contents of the receipt as fields	• TaxpayerManager • Taxpayer • Company

Class Name: Company	
Responsibilities	**Collaborations**
• Keeps company information that create the receipt as fields	• TaxpayerManager • Taxpayer • Receipt

Class Name: FactoryTaxPayer

Responsibilities	Collaborations
• Returns the right category of the taxpayer • Returns all values that this category needs	

Class Name: FactoryFile	
Responsibilities	**Collaborations**
• Returns the boolean if the type is txt • Returns the boolean if the type is xml	• TaxpayerManager • FactoryFileWriter • FactoryFileReader

Class Name: FactoryFileWriter	
Responsibilities	**Collaborations**
• Returns a filewriter txt or xml object with with all information	• TaxpayerManager • FactoryFile

Class Name: FactoryFileReader	
Responsibilities	**Collaborations**
• Returns a filereader xml or txt object with with all information	• TaxpayerManager • FactoryFile

- **<u>Package :</u> incometaxcalculator.data.io**

Class Name: FileReader	
Responsibilities	**Collaborations**
• Define the function checkForReceipt() • Define the function getValueOfField()	• TXTFileReader • XMLFileReader

Class Name: TXTFileReader	
Responsibilities	**Collaborations**
• Create the body of checkForReceipt () function in order to obtain the field proof value from the txt file • Create the body of getValueOfField() function to get the values from all the fields of receipt	• FileReader

Class Name: XMLFileReader	
Responsibilities	**Collaborations**
• Create the body of checkForReceipt () function in order to obtain the field proof value from the xml fil • Create the body of getValueOfField() function to get the values from all the fields of receipt	• FileReader

Class Name: FileWriter	
Responsibilities	**Collaborations**
• Defines the generateFile() function • Represents the functions that the classes inherited from them can implement in their own way.	• TXTInfoWriter • XMLInfoWriter • TXTLogWiter • XMLLogWiter • LogWriter • InfoWriter

Class Name: LogWriter	
Responsibilities	**Collaborations**

• Writes the right form of log txt files • Writes the right form of log xml files	• FileWriter • TXTLogWiter • XMLLogWiter

Class Name: InfoWriter	
Responsibilities	**Collaborations**
• Writes the right form of log txt files • Writes the right form of log xml files	• FileWriter • TXTInfoWiter • XMLInfoWiter

Class Name: TXTInfoWriter	
Responsibilities	**Collaborations**
• Returns the begin tags of each form • Returns the end tags of each form	• FileWriter • InfoWriter

Class Name: XMLInfoWriter

Responsibilities	Collaborations
• Returns the begin tags of each form • Returns the end tags of each form	• FileWriter • InfoWriter

Class Name: XMLLogWriter	
Responsibilities	**Collaborations**
• Returns the begin tags of each form • Returns the end tags of each form	• FileWriter • Logwriter

Class Name: TXTLogWriter	
Responsibilities	**Collaborations**
• Returns the begin tags of each form • Returns the end tags of each form	• FileWriter • Logwriter

EXPLANATION THE WAY OF SOLVING PROBLEMS

GENERAL PROBLEMS

incometaxcalculator.data.management package:

1. Date class:

The problem here was Unnecessary Complexity. Java provides its own Calendar object that is used instead of this Date class.

2. Company class:

The problem here was Unnecessary Complexity. The Company class contains dead code and Address class that was removed .

3. Taxpayer class:

addReceipt(), removeReceipt(), getVariationTaxOnReceipts() have some complex conditional logic in the form of chained if-else statements. Simplify these methods by changing the algorithm. The idea was to use a for loop instead of the chained if-else statements.

4.Subclasses of the Taxpayer class:

The problem here was Duplicate Code. Observe that the calculateBasicTax() method in every subclass is similar. All the methods perform the same steps which differ only in some constant values. So we removed the code duplication using parameterization.

Our idea here is that we use a couple of simple data structures (hashmap and arrays) as fields in the Taxpayer class for storing the different constants. Then, change the calculateBasicTax() body to use these data structures instead of the constants. This will result in having the same method every subclass, which is pulled up to the base class. The subclasses are used just to initialize the values of the simple data structures in the respective constructors. Alternatively, the subclasses were removed; in this case the initialization of the data structures could be done using respective property configuration files.

5. TaxpayerManager class:

This is a Large Class with many responsibilities. Simplify this class by delegating some responsibilities to subordinate classes:

a. createTaxpayer(): you moved the conditional logic that creates different types of Taxpayer objects to a simple parameterized factory.

b. updateFiles(): you moved common parts out of the complex if-else logic; then you moved the conditional logic that creates different types of FileWriter objects to a simple parameterized factory.

c. saveLogFile(): you moved common parts out of the complex if-else logic; then you moved the conditional logic that creates different types of FileWriter objects to a simple parameterized factory.

d. loadTaxpayer(): you moved common parts out of the complex if-else logic; then you can move the conditional logic that creates different types of FileReader objects to a simple parameterized factory.

incometaxcalculator.data.io package:

1.TXTFileReader, XMLFileReader classes:

The problem here was Duplicate Code. The core algorithms of the classes methods are similar. We removed the code duplication. The idea here is to form template methods in the FileReader super class and abstract the parts of the code that are different with simple abstract methods implemented in the subclasses.

2. FileWriter class:

One problem with this class is that it is a Middle Man for TaxpayerManager; it had many methods that simply delegate calls to respective methods of TaxpayerManager Another problem was Refuse Bequest, i.e. some methods (e.g., getReceipt*() methods, getCompany*() methods) were used only by some of the subclasses. An idea here was to simplify: we removed the delegating methods and called directly the TaxpayerManager methods; pushed down methods to the classes that need them; made FileWriter an abstract instead of an interface class.

3. TXTInfoWriter and XMLInfoWriter classes:

The problem here was Duplicate Code. The core algorithms of the classes methods were similar. They only differ in constant string tags that were written along with the basic information. So we removed the code duplication. The idea here is to form template methods in an abstract InfoWriter super class (that implements the FileWriter abstract) and abstract the parts of the code that were different with simple abstract methods extended in the subclasses. The different extentations of the simple abstract methods return different string constants.

4. TXTLogWriter and XMLLogWriter classes:

Again the problem here was Duplicate Code. The core algorithms of the classes methods were similar. They only differ in constant string tags that were written along with the basic information. So we removed the code duplication. The idea here is to form template methods in an abstract InfoWriter super class (that implements the FileWriter abstract) and abstract the parts of the code that were different with simple abstract methods extended in the subclasses. The different extentations of the simple abstract methods return different string constants.