REFACTORING
SALES & COMMISSIONS MANAGEMENT APPLICATION

OVERALL REPORT

VERSION <1.1>

<Theodoros Florentzis AM: 4517>
<Dimitrios Gazos AM:4035>

TABLE OF CONTENTS

INTRODUCTION

This project focuses thoroughly on refactoring and re-engineering a Java application designed for managing sales and commissions to an agent by given receipts within a small clothing company. Practical usage of several techniques and practices for clean software development are used such as:

- Naming conventions, code organization and error handling
- Refactoring techniques (composing, simplifying and moving methods)
- Design patterns

Also new features where added to the application, like importing and exporting formatted HTML files.

This report serves as a comprehensive guide to the changes we were requested to implement to the project in order to enhance its efficiency, code clarity, maintainability and scalability.

The project was implemented on *Eclipse*. For the UML diagrams we used an older installation of ObjectAid. Any changes on the GUI of the application where done using the *Eclipse Windowbuilder plugin.*

REFACTORED DESIGN

USE CASES

- InputAgentDataFromTXT

Use case ID	UC1
Actors	Agent, Company employer
Pre conditions	The agent must have a specific formatted file {.txt} containing his name, AFM and receipts history.
Main flow of events	1. The use case starts when the user selects the “Εισαγωγή από TXT” button. 2. The application opens a file explorer window. 3. The user selects the desired .txt file and presses the “open” button.
Alternative flow 1	1. The user can exit the use case at any point by closing the file explorer window. 2. The application shows an appropriate message that no file was selected.
Alternative flow 2	1. One or more fields in the file have wrong type of value (ex. int instead of string). 2. The application shows an appropriate message that there was an error parsing the file.
Alternative flow 3	1. The user selects the same receipts data file again. 2. The application shows an appropriate message that the agent is already loaded in the application.
Post conditions	1. The agent’s data has been loaded to the application. 2. The agent has been added to the list of representatives.

- InputAgentDataFromXML

Use case ID	UC2
Actors	Agent, Company employer
Pre conditions	The agent must have a specific formatted file {.xml} containing his name, AFM and receipts history.
Main flow of events	1. The use case starts when the user selects the “Εισαγωγή από XML” button. 2. The application opens a file explorer window. 3. The user selects the desired .xml file and presses the “open” button.
Alternative flow 1	1. The user can exit the use case at any point by closing the file explorer window. 2. The application shows an appropriate message that no file was selected.
Alternative flow 2	1. One or more fields in the file have wrong type of value (ex. int instead of string). 2. The application shows an appropriate message that there was an error parsing the file.
Alternative flow 3	1. The user selects the same receipts data file again. 2. The application shows an appropriate message that the agent is already loaded in the application.
Post conditions	1. The agent’s data has been loaded to the application. 2. The agent has been added to the list of representatives.

- InputAgentDataFromHTML

Use case ID	UC3
Actors	Agent, Company employer
Pre conditions	The agent must have a specific formatted file {.html} containing his name, AFM and receipts history.
Main flow of	1. The use case starts when the user selects the “Εισαγωγή από HTML” button. 2. The application opens a file explorer window.

events	3. The user selects the desired .html file and presses the “open” button.
Alternative flow 1	1. The user can exit the use case at any point by closing the file explorer window. 2. The application shows an appropriate message that no file was selected.
Alternative flow 2	1. One or more fields in the file have wrong type of value (ex. int instead of string). 2. The application shows an appropriate message that there was an error parsing the file.
Alternative flow 3	1. The user selects the same receipts data file again. 2. The application shows an appropriate message that the agent is already loaded in the application.
Post conditions	1. The agent’s data has been loaded to the application. 2. The agent has been added to the list of representatives.

- SelectAgent

Use case ID	UC4
Actors	Agent, Company employer
Pre conditions	The user must have executed either UC1 or UC2 or UC3.
Main flow of events	1. The use case starts when the user starts the application. 2. The user selects the agent’s name from the agents’ list. 3. The user presses the “ok” button.
Alternative flow 1	1. The user does not select an agent and presses the “ok” button. 2. The application shows an appropriate message that no agent is selected.
Alternative flow 2	The user can at anytime close the application by pressing the “cancel” button or by closing the window.
Alternative flow 3	The user can at any time execute either UC1 or UC2 or UC3 again.

Post conditions	1. The application changes to another interface. 2. The receipts data from the selected agent are available to the user for further processing.

- SelectDataForPresentation

Use case ID	UC5
Actors	Agent, Company employer
Pre conditions	The user must have executed UC4.
Main flow of events	1. The use case starts right after the execution of UC4. 2. The application displays: 2.1. The selected agent's name. 2.2. Four check-box buttons regarding information to be displayed about the agent 2.3. Four radio buttons included in the "Πωλήσεις συγκεκριμένης κατηγορίας' button. 3. The user checks one or more of the check-boxes. 4. If "Πωλήσεις συγκεκριμένης κατηγορίας" button is selected then 4.1. The user selects one or more of the radio buttons. 5. The user presses the "ok" button. 6. The application changes to a new interface displaying the information that the user selected.
Alternative flow 1	1. The user presses the "cancel" button. 2. The application returns to the previous interface.
Alternative flow 2	The user at any time can close the application by closing the window.
Post conditions	The application changes to a new interface.

- AddReceipt

Use case ID	UC6
Actors	Agent
Pre conditions	The user must have executed UC4.
Main flow of events	1. The use case starts when the user presses the “Προσθήκη νέας απόδειξης” button. 2. The application shows several text fields to be completed from the user regarding data for the new receipt. 3. While the user doesn’t press again the “Προσθήκη νέας απόδειξης” button 3.1. The user inputs the desired information to each text field. 3.2. The user presses the “Προσθήκη” button at the bottom. 3.3. The application shows a message that the receipt has been added successfully. 3.4. The counter field “Αποδείξεις που προστέθηκαν”, at the lower bottom of the window, increases by one.
Alternative flow 1	1. The user presses the “cancel” button. 2. The application returns to the previous interface.
Alternative flow 2	The user at any time can close the application by closing the window.
Alternative flow 3	1. The user doesn’t fill any of the text fields and presses the “Προσθήκη” button. 2. The application shows an appropriate message of failure that some text fields are empty.
Alternative flow 4	1. The user doesn’t fill one or more of the text fields or inputs wrong type of data and presses the “Προσθήκη” button. 2. The application shows an appropriate message of failure that some text fields are empty.
Post conditions	The receipts that the user added have been saved to the agent’s data file.

- ExportToTXT

Use case ID	UC7
Actors	Agent, Company employer
Pre conditions	The agent must have executed UC5.
Main flow of events	1. The use case starts when the user presses the “Εξαγωγή σε TXT” button. 2. The application opens a save dialog window. 3. The user selects the desired path and name of the export {.txt} file. 4. The user presses the “save” button. 5. The application shows a success message.
Alternative flow 1	The user at any time can exit the use case by closing the save dialog or by pressing the “cancel” button.
Post conditions	The TXT report file has been created.

- ExportToXML

Use case ID	UC8
Actors	Agent, Company employer
Pre conditions	The agent must have executed UC5.
Main flow of events	1. The use case starts when the user presses the “Εξαγωγή σε XML” button. 2. The application opens a save dialog window. 3. The user selects the desired path and name of the export {.xml} file. 4. The user presses the “save” button. 5. The application shows a success message.

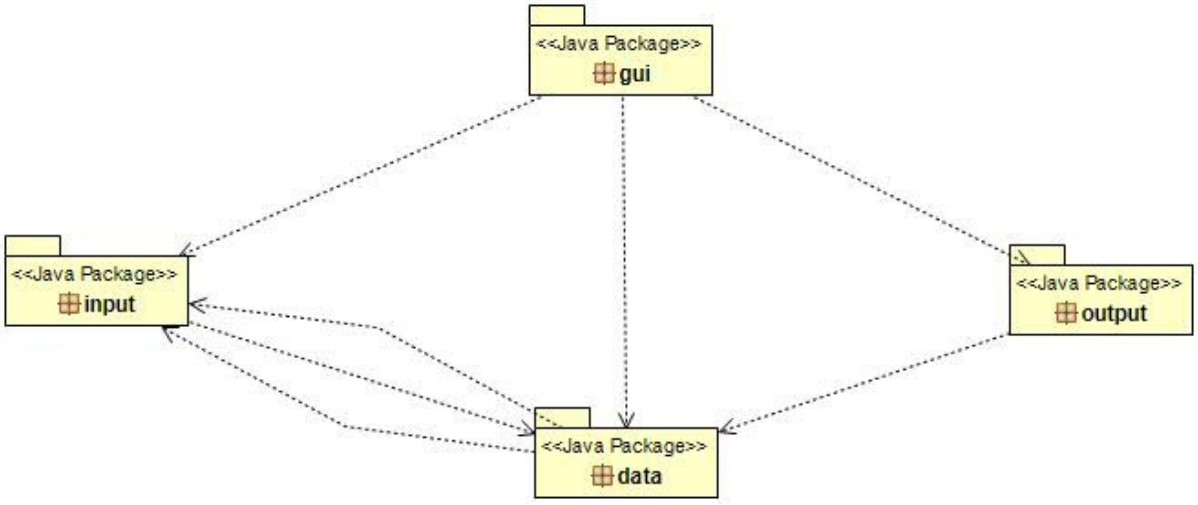
Alternative flow 1	The user at any time can exit the use case by closing the save dialog or by pressing the “cancel” button.
Post conditions	The XML report file has been created.

- ExportToHTML

Use case ID	UC9
Actors	Agent, Company employer
Pre conditions	The agent must have executed UC5.
Main flow of events	1. The use case starts when the user presses the “Εξαγωγή σε HTML” button. 2. The application opens a save dialog window. 3. The user selects the desired path and name of the export {.html} file. 4. The user presses the “save” button. 5. The application shows a success message.
Alternative flow 1	The user at any time can exit the use case by closing the save dialog or by pressing the “cancel” button.
Post conditions	The HTML report file has been created.

ARCHITECTURE

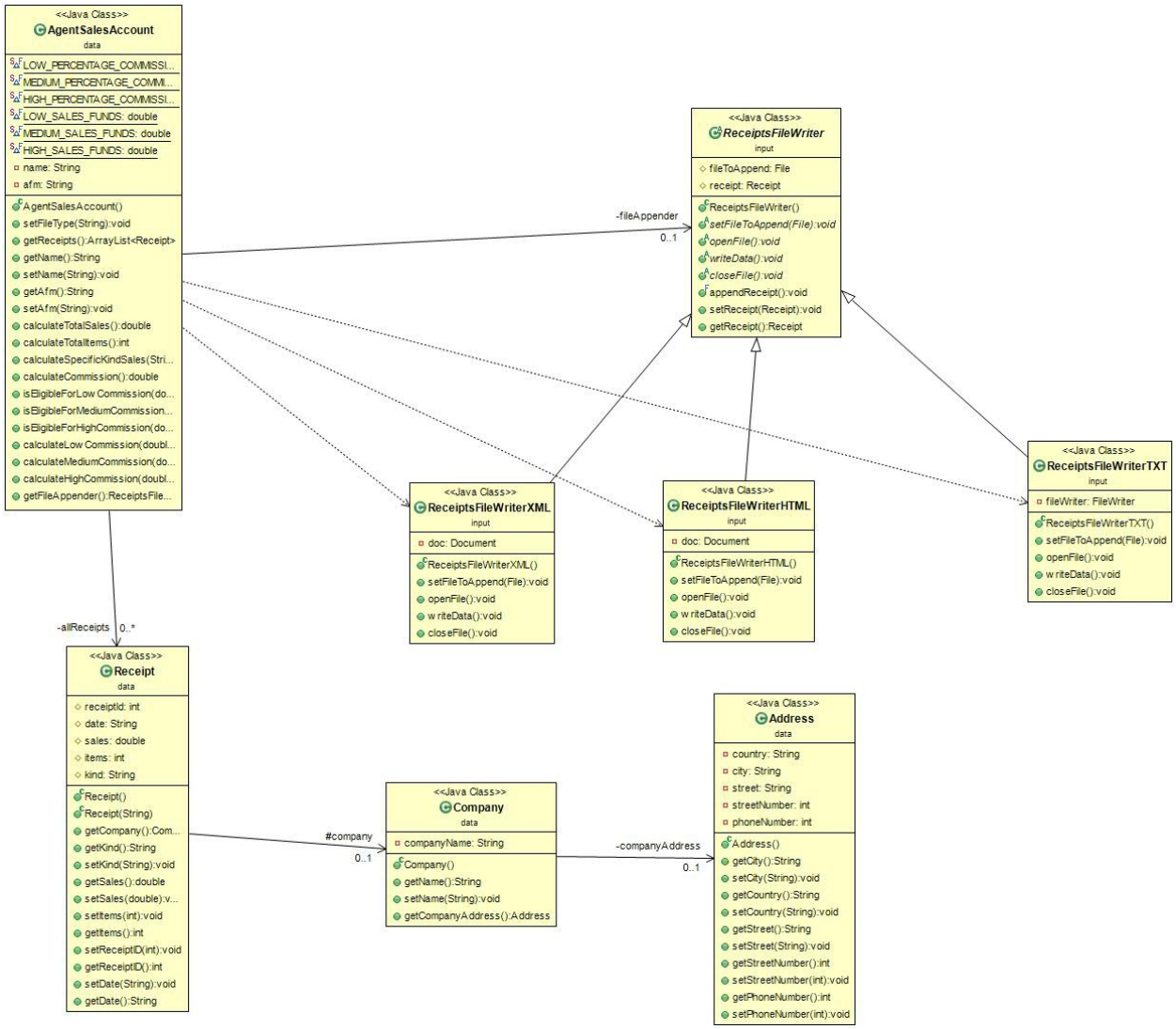
- The overall UML package diagram that shows the architecture of the **refactored** application.



DETAILED DESIGN

In this section of the report we provide the UML class diagrams corresponding to each package of the refactored application. Also we examine in detail the changes we made to the application in order to meet the refactoring requirements listed on the given backlog.

- **DATA PACKAGE**



1. *Coat, Shirt, Skirt, Trouser classes: the problem here is Lazy Classes that do not do much to deserve to exist. The lazy classes can be removed.*

- Added the method isSpecificKind() in input/input class. This boolean method returns true if given kind value is equal with one of the four specified items in the store.

- Added new constructor in data/Receipt class to handle and create Receipt object with “kind” argument. There is also another constructor that does not require the “kind” argument and set the “kind” field of the object as “No specific kind”.

- Changes to the addReceipt() method in Input class. With use of isSpecificKind(), if the given kind from input equals with one of the four specified items, create a new receipt with the kind argument. Otherwise it creates a new receipt with “No specific kind”.
  Same changes to the same method in gui/AgentOperationsWindow class when the user inputs “kind” type in gui.

- Removed: The four lazy classes as we handle the kind as an argument instead of calling it as an individual class.

2.a. *The name of the Agent class does not reflect the role of the class in the application domain.*

- Changed the name of Agent class to AgentSalesAccount in order to better specify the role of the class that calculates sales and commissions made by an agent.

2.b. *Several methods in Agent class differ only in constants and variables. The duplicate code should be removed.*

- Removed: The four methods calculate<kind>Sales() (one of each specified item) as the were very similar (only kind value changed).

- Created a new method calculateSpecificKindSales(String kind) to calculate each item sales depending on the given kind argument. This method replaces the four methods we removed earlier.

- Changed the method used to calculate each specific kind in output/ReportCreationTXT class to the new calculateSpecificKindSales(String kind) we created. Same to output/ReportCreationXML class.

- Changed the different commission percentages from plain numbers to symbolic constants to be more comprehensive. Same for the numbers that represent the boundaries between sales made for the commission. (Symbolic Constant Method)

- Added local variable totalSales inside calculateCommission() method to calculate total sales of agent instead of calling the calculateTotalSales method 8 times.

- Created 3 new methods isEligibleFor{Low/Medium/High}Commission replacing the long parameters in calculateCommission() method, that checks how much commission percentage the agent is eligible for.

- Created 3 new methods calculate{Low/Medium/High}Commission replacing the long parameters in calculateCommission() method, that calculate the total commission for the agent depending on the sales.

2.c. *The class is using the Java Vector data structure which is deprecated.*

- Replaced <vector> data type with ArrayList data type.

3.a. *The classes FileAppender, FileAppenderTXT, FileAppenderXML update the contents of files that contain receipt information. Based on this functionality the names of the classes are not clear.*

- Changed the name of FileAppender class to ReceiptsFileWriter, FileAppenderTXT class to ReceiptsFileWriterTXT, FileAppenderXML class to ReceiptsFileWriterXML in order to better specify the roles of these classes that write receipts on files.

3.b. *Since the classes perform IO operations it seems that they are placed in the wrong package.*

- Transferred the files from data package to input package because the user inputs data to a file. Eclipse was used to transfer the files to the new package.

4. *The class ReceiptsFileWriter has too many fields of a primitive type that can be replaced by a Receipt object that encapsulates the primitive data.*

- Removed: The 10 protected strings fields of the class and replaced them with a Receipt object as a new field.

- Removed: The 10 field sets.

- Added a set method for the new Receipt object field of the class. The new receipt object is created in the AgentOperationsWindow class, so we needed a set method to pass the Receipt object to the ReceiptsFileWriter class.

- We access the values we need thru the Receipt object field we created using the get methods that are implemented in the Receipt class. For example receipt.getReceiptID().

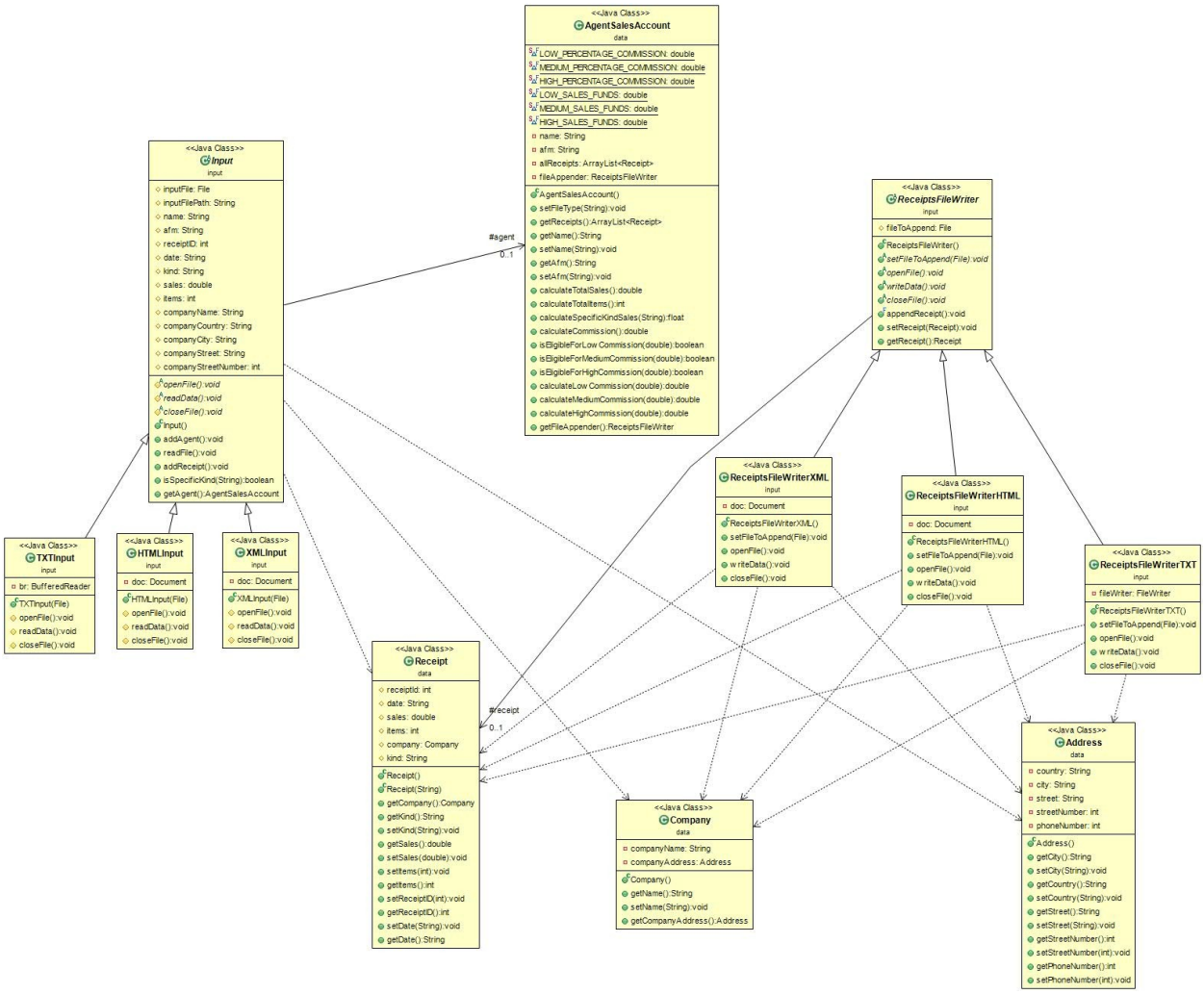
  The above have been implemented for both ReceiptsFileWriterTXT and ReceiptsFileWriterXML.

5. *The common algorithm in classes ReceiptsFileWriterTXT and ReceiptsFileWriterXML can be extracted in a template method that will be placed in the ReceiptsFileWriter class.*

- At appendReceipt() method in ReceiptsFileWriter abstract class we created a template openFile()/writeData()/closeFile() as both of ReceiptsFileWriterTXT and ReceiptsFileWriterXML have this code structure as common. (Template Method)

- In ReceiptsFileWriterTXT and ReceiptsFileWriterXML we moved and grouped the code into the three methods we created in the super class appendReceipt() to match the same algorithmic steps from template.

◆ INPUT PACKAGE



1. *The algorithm that is used in the TXTinput class to parse txt input files is more complex than necessary given that the order of the elements within the txt file is pre-determined and fixed.*

- Removed: the if statements since the format is already known and there is no need to check.

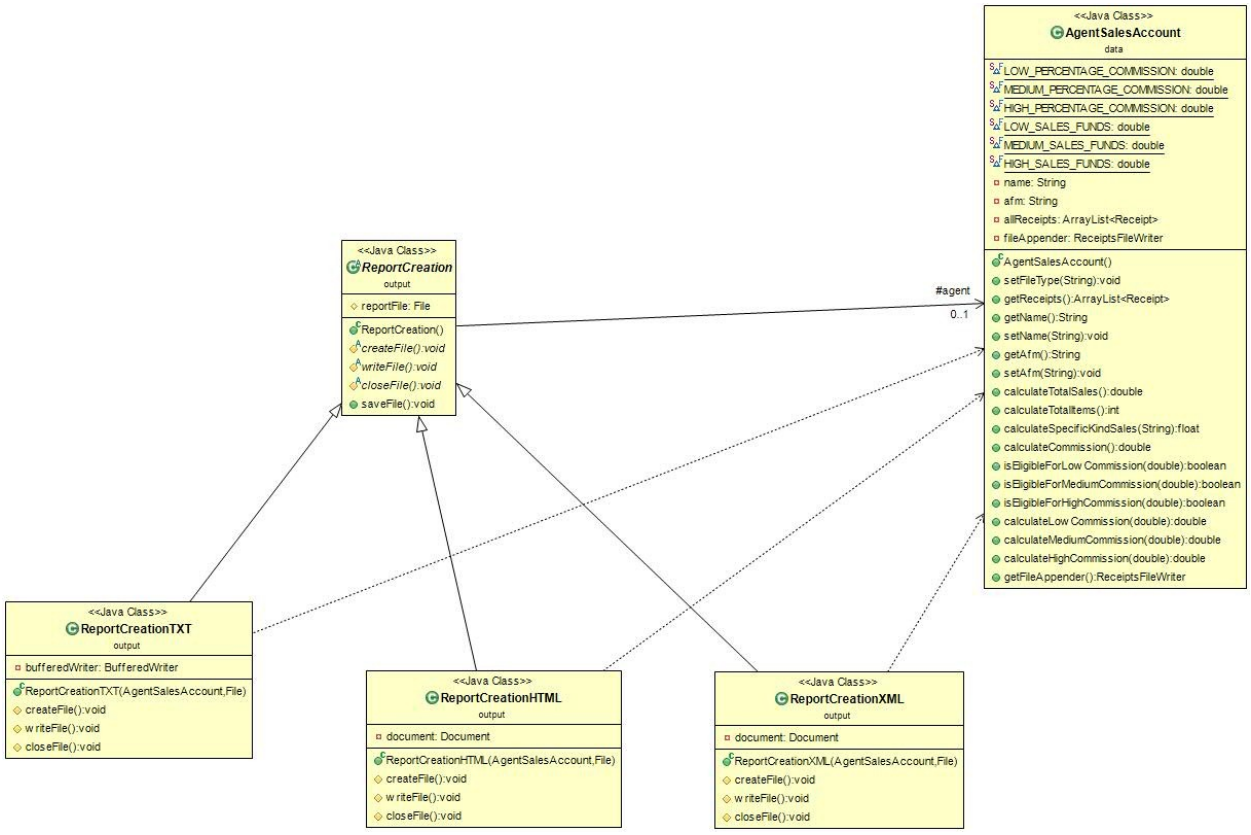
- We simplified the application's reading .txt file process by modifying it to read the file line by line with br.readLine() in alignment with correct spaces.

2. *After the refactoring of TXTInput to a simpler algorithm, chances are that the parsing algorithms of TXTInput and XMLInput would follow the same steps. The common algorithm can be extracted in a template method that will be placed in the Input class.*

- At readFile() method in Input abstract class we created a template openFile()/ReadData()/closeFile() as both of TXTInput and XMLInput have this code structure as common after the refactoring. (Template Method)

- In TXTInput and XMLInput we moved and grouped the code into the three methods we created in the super class readFile() method to match the same algorithmic steps from template.

◆ OUTPUT PACKAGE



***1*.a.** *The names of the classes Report, TXTReport and XMLReport are not adequate. Observe that the classes generate reports they are not the reports themselves. Better names should be given.*

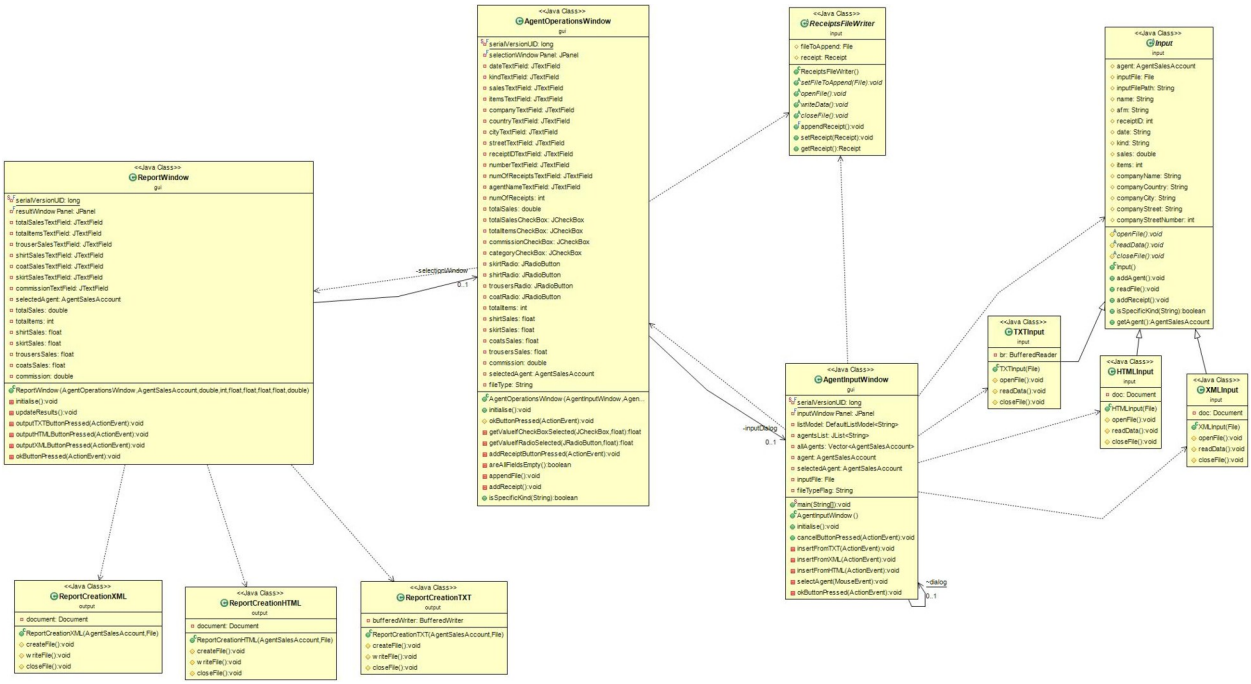
- Changed the name of Report class to ReportCreation, TXTReport class to ReportCreationTXT and XMLReport class to ReportCreationXML in order to better specify the role of creating the output report.

***1*.b.** *The common algorithm can be extracted in a template method that will be placed in the base class.*

- In saveFile() method in ReportCreation abstract class we created a template createFile()/writeData()/closeFile() as both of ReportCreationTXT and ReportCreationXML have this code structure as common. (Template Method)

- In ReportCreationTXT and ReportCreationXML we moved and grouped the code into the three methods we created in the super class saveFile() method to match the same algorithmic steps from template.

- **GUI PACKAGE**



For modifying and expanding the gui of the application we used the Eclipse Windowbuilder plugin.

1. *Refactor the gui to allow the user select a particular file for saving a report, instead of using a predefined file name*

- Modified the method outputTXTButtonPressed() so that the user chooses the name and path of the exported report by using JFileChooser and .showSaveDialog. We save the file using .getSelectedFile to a File constant. Same technique was used in outputXMLButtonPressed. We replace the static fullPathFile string variable as we now choose the path.

- Modified ReportCreationTXT and ReportCreationXML arguments to pass the new created file as an argument.

2. *The names of the classes InputWindow, SelectionWindow and ResultWindow can be improved to better reflect their responsibilities.*

- Changed the name of InputWindow class to AgentInputWindow to specify the interface for adding agent profiles. SelectionWindow class to AgentOperationsWindow to show what kind of operations the user is allowed to execute. Lastly changed the ResultWindow class to ReportWindow in order to specify the interface that the user exports the report.

3**.a.** *The addReceiptButtonPressed() in AgentOperationsWindow class has a very complex conditional statement that can be improved.*

- Created method in AgentOperationsWindow class areAllFieldsEmpty(). We extract the if parameter into the method areAllFieldsEmpty() that returns true if the user has left all text fields empty. (Replace long parameter with method technique)

3**.b.** *In the appendFile() method in AgentOperationsWindow class we observe chains of method calls that can be removed. The size of the method can also be significantly reduced based on the data package refactoring 4.*

- We observed that the receipt object is created before we write it to the receipts file. So now we obtain the last element from allReceipts field of the AgentSalesAccount class and we set the newly created receipt field of the ReceiptsFileWriter class. With this way, we are not obliged to set each value individually but rather we set them as an object.

- Added the method isSpecificKind(). This boolean method returns true if given kind value is equal with one of the four specified items in the store. So in the addReceipt method, to obtain the string value of the JTextField, we use the .getText() method.

***3*.c.** *The code of the okButtonPressed() method in AgentOperationsWindow class seems more complicated than necessary. Probably the multiple if-else statements can be simplified.*

- Created two new methods getValueIfCheckBoxSelected() and getValueIfRadioSelected() where they are responsible to check if a check-box or a radio-box is pressed by the user. If a check-box or a radio-box is selected, then they return the according calculation.

- Replaced the if-else statements inside okButtonPressed() and we use the methods getValueIfCheckBoxSelected() and getValueIfRadioSelected() to calculate the values we need selected by the user.

■ EXTENSION TASK

***1*.** *Provide input to the application via an html file. The different elements of the file should be structured with respect to appropriate html tags like headings, lists and so on.*

- Added parser JSOUP in our eclipse project in order to handle the known html tags. We include the parser to our deliverable project files.

- Created a new test input file (test-case-3-HTML.html) structured using only HTML tags. This file contains agent's data and receipts. Also we improved the style of the html page using <style> tag in order to be better presentable by adding colors, changing letter sizes etc.

- Created a new HTMLInput class in input package. This class is responsible for reading a HTML file and extract agent's data and receipts.

  Inside each tag we save individual receipts. We use it to distinguish each receipt. The way we gather the receipt's information inside the tags is similar to XML but with a few differences. For example:

name = nodeLst.first().select("h2").text().trim().replace("Name: ", "");

- *nodeLst* → Contains the <body> tag in a list with one element.
- *First()* → We take the one and only element.
- *Select("h2").text()* → We take the text of the <h2> tag.
- *replace("Name: ","")* → We keep only the value of the name field.

- Added a button *"Εισαγωγή από HTML"* in the GUI interface in gui/AgentInputWindow class. This button is pressed by the user in order to input a formatted html file to the application.

- Added a new method insertFromHTML() in AgentInputWindow class. Similar function with insertFromXML class but with the appropriate name changes. It is used to load a HTML file to our application by calling the proper methods we created in the back-end.

- Created new input/ReceiptsFileWriterHTML class to add receipts to the chosen agent's file by creating and appending a element, to the agent's file, containing multiple <p> tags with the appropriate receipt values.

2. *Store a report for a sales representative in an html file. The different elements of the file should be structured with respect to appropriate html tags like headings, lists and so on.*

- Added in the if-statements of the setFileType() Method in data/AgentSalesAccount class an else statement that covers the case of an HTML file.

- Created new output/ReportCreationHTML class to export the report in a HTML format. Also added the style of the input test case html file to be more presentable.

- Added a button *"Εξαγωγή σε HTML"* in the GUI interface in gui/ReportWindow class. The user presses the button when he wants to export the agent's data in a HTML formatted report.

■ ADDITIONAL CHANGES

- Updated the text in error message dialogs to now display error messages written in Greek language. (ISO 8859-7)

CLASSES RESPONSIBILITIES AND COLLABORATIONS (CRC CARDS)

◆ DATA PACKAGE

Class Name: Address	
Responsibilities	**Collaborations (classes)**
▪ This class is responsible for implementing an address object in our application.	▪ Company

Class Name: AgentSalesAccount	
Responsibilities	**Collaborations (classes)**
▪ This class is responsible for implementing an Agent object in our application.	▪ Receipt ▪ ReceiptsFileWriter ▪ ReceiptsFileWriterTXT ▪ ReceiptsFileWriterXML

▪ Calculates the following regarding the Agent: o Commission o Total sales o Total items o Specific kind items	▪ ReceiptsFileWriterHTML

Class Name: Company	
Responsibilities	**Collaborations (classes)**
▪ This class is responsible for modeling a company object in our application.	▪ Address ▪ Receipt

Class Name: Receipt	
Responsibilities	**Collaborations (classes)**
▪ This class is responsible for creating a receipt object in our application.	▪ AgentSalesAccount ▪ Company

◆ INPUT PACKAGE

Class Name: Input	
Responsibilities	**Collaborations (classes)**
▪ This abstract class is responsible for the functionality of loading the agent's data into the application. ▪ Setting up the template to extract the common algorithm in child classes.	▪ TXTInput ▪ XMLInput ▪ HTMLInput ▪ Receipt ▪ Company ▪ Address ▪ AgentSalesAccount

Class Name: TXTInput	
Responsibilities	**Collaborations (classes)**
▪ This class is responsible for reading the Agent's data from a TXT format file.	▪ Input

Class Name: XMLInput	
Responsibilities	**Collaborations (classes)**
▪ This class is responsible for reading the Agent's data from an XML format file.	▪ Input

Class Name: HTMLInput	
Responsibilities	**Collaborations (classes)**
▪ This class is responsible for reading the Agent's data from a HTML format file.	▪ Input

Class Name: ReceiptsFileWriter	
Responsibilities	**Collaborations (classes)**
▪ This abstract class is responsible for the functionality of appending receipts to a file. ▪ Setting up the template to extract the common algorithm in child classes.	▪ ReceiptsFileWriterTXT ▪ ReceiptsFileWriterXML ▪ ReceiptsFileWriterHTML ▪ Receipt

Class Name: ReceiptsFileWriterTXT	
Responsibilities	**Collaborations (classes)**
▪ This class is responsible for appending a receipt to the selected TXT file by the user.	▪ ReceiptsFileWriter ▪ Address ▪ Company ▪ Receipt

▪ The user must have selected an agent's file with TXT format.	

Class Name: ReceiptsFileWriterXML	
Responsibilities	**Collaborations (classes)**
▪ This class is responsible for appending a receipt to the selected file by the user. ▪ The user must have selected an agent's file with XML format.	▪ ReceiptsFileWriter ▪ Address ▪ Company ▪ Receipt

Class Name: ReceiptsFileWriterHTML	
Responsibilities	**Collaborations (classes)**
▪ This class is responsible for appending a receipt to the selected file by the user. ▪ The user must have selected an agent's file with HTML format.	▪ ReceiptsFileWriter ▪ Address ▪ Company ▪ Receipt

- **OUTPUT PACKAGE**

Class Name: ReportCreation	
Responsibilities	**Collaborations (classes)**
▪ This abstract class is responsible for the functionality of creating reports in different formats (TXT/XML/HTML). ▪ Setting up the template to extract the common algorithm in child classes.	▪ ReportCreationTXT ▪ ReportCreationXML ▪ ReportCreationHTML ▪ AgentSalesAccount

Class Name: ReportCreationTXT	
Responsibilities	**Collaborations (classes)**
▪ this class is responsible for exporting a selected agent's report in a TXT format.	▪ ReportCreation ▪ AgentSalesAccount

Class Name: ReportCreationXML	
Responsibilities	**Collaborations (classes)**
▪ this class is responsible for exporting a selected agent's report in an XML format.	▪ ReportCreation ▪ AgentSalesAccount

Class Name: ReportCreationHTML	
Responsibilities	**Collaborations (classes)**
▪ this class is responsible for exporting a selected agent's report in a HTML format.	▪ ReportCreation ▪ AgentSalesAccount

◆ GUI PACKAGE

Class Name: AgentInputWindow	
Responsibilities	**Collaborations (classes)**
▪ This class is responsible for providing the graphical interface when the user launches the application. ▪ This interface provides ways to add files in several formats and the list of added agents for the user to select.	▪ AgentOperationsWindow ▪ ReceiptsFileWriter ▪ Input ▪ TXTInput ▪ XMLInput ▪ HTMLInput

Class Name: AgentOperationsWindow	
Responsibilities	**Collaborations (classes)**
▪ This class is responsible for providing the graphical interface showing operations the user can execute. ▪ This interface displays check boxes that the user can check to display several information and way to add receipts to the agent's file.	▪ AgentInputWindow ▪ ReceiptsFileWriter ▪ ReportWindow

Class Name: ReportWindow	
Responsibilities	**Collaborations (classes)**
▪ This class is responsible for providing the graphical interface showing selected data and file types to export the agent's report. ▪ This interface displays data the user selected in the previous interface to show and buttons to select file types to export an agent's report.	▪ AgentOperationsWindow ▪ ReportCreationTXT ▪ ReportCreationXML ▪ ReportCreationHTML