<SALES-COMMISSIONS APPLICATION>

OVERALL REPORT

VERSION <1.0>

<Alexandros-Dimitrios Tzimogiannis, A.M.: 4179

Georgios Triantos, A.M.: 4184>

TABLE OF CONTENTS

INTRODUCTION

This document provides information concerning the final sprint of the project. This app was refactored for educational purposes under the class Software Development II MYE004, taught by Apostolos Zarras at the Department of Computer Science and Engineering, UoI.

REFACTORED DESIGN

USE CASES

1. <IMPORTFILE>

Use case ID	UC1
Actors	Ordinary User
Pre conditions	1. The application is up and running. 2. The application supports at least .txt, .xml and .html file formats.
Main flow of events	1. The use case starts when the user presses the “Εισαγωγή από TXT” or “Εισαγωγή από XML” or “Εισαγωγή από HTML” button. 2. The system opens a fileChooser. 3. The user chooses a file to open (txt, xml or html). 4. The app presents the name of the agents listed in the chosen file.
Alternative flow	The user presses the Cancel button and the application closes.
Post conditions	The agents’ names of the file that the user imported have been displayed.

2. <SELECTAGENT>

Use case ID	UC2
Actors	Ordinary User
Pre conditions	1. The application is up and running. 2. There is at least one agent name at the agents list(“Λίστα Αντιπρσώπων”).
Main flow of events	1. The use case starts when the user selects a certain agent. 2. The user presses the OK button.
Alternative flow	1. The user presses the Cancel button and the application closes.
Post conditions	The application opens a new window to select certain informations about the agent.

3. <SALESFILTER>

Use case ID	UC3
Actors	Ordinary User
Pre conditions	1. The application is up and running. 2. The user has selected an agent from the list.
Main flow of events	1. The use case starts when the new window presents the available filters for the user. 2. The user selects the preferred filters. 3. The user presses the OK button.
Alternative flow	1. The user presses the Cancel button the application reopens the first window.
Post conditions	The application opens a new window with the selected information.

4. <ADDRECEIPT>

Use case ID	UC4
Actors	Ordinary User
Pre conditions	1. The application is up and running. 2. The user has selected an agent from the list.
Main flow of events	1. The use case starts when the user presses the "Προσθήκη νέας απόδειξης" button . 2. The user writes the information of the receipt. 3. The user presses the button "Προσθήκη".
Alternative flow	1. The user presses the Cancel button the application reopens the first window.
Post conditions	The new receipt is stored in the previously imported file.

5. <SAVEREPORT>

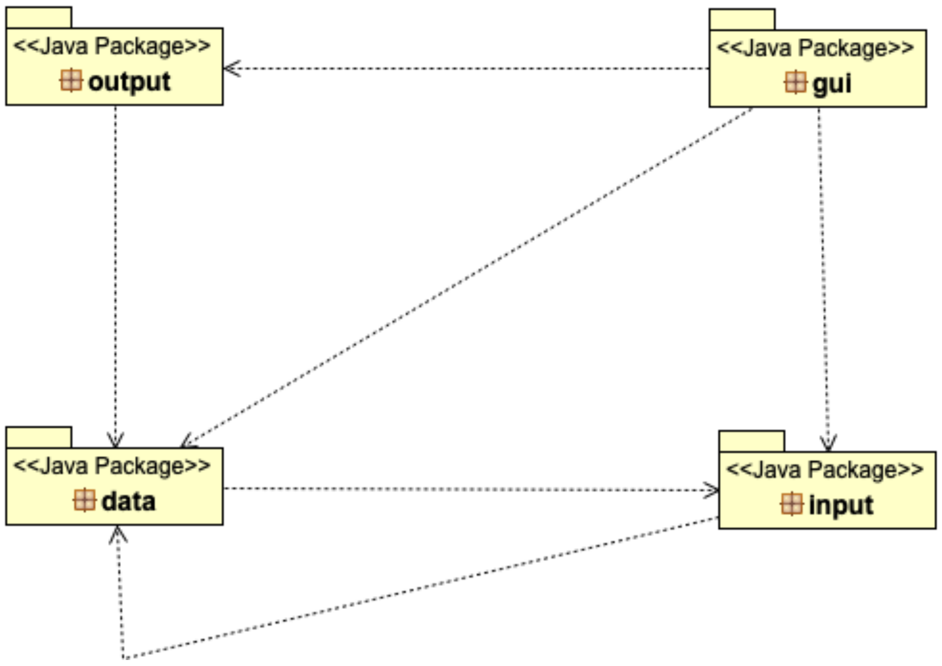
Use case ID	UC5
Actors	Ordinary User
Pre conditions	1. The application is up and running. 2. The application has opened the receipts information window.
Main flow of events	1. The use case stars when the user presses the "Εξαγωγή από TXT" or "Εξαγωγή από XML" or "Εξαγωγή από HTML" button. 2. The system opens a fileChooser. 3. The user chooses a file to store the report (.txt, .xml, .html).
Alternative flow 1	1. At the fileChooser the user stores the report at a new file.
Alternative flow 2	The user presses the Cancel button the application reopens the previous window.
Post conditions	The report has been stored in the selected file.

6. <EXITAPP>

Use case ID	UC6
Actors	Ordinary User
Pre conditions	1. The application is up and running. 2. The application has opened the receipts information window.
Main flow of events	1. The use case stars when the user presses the OK button.
Post conditions	The application has been terminated.

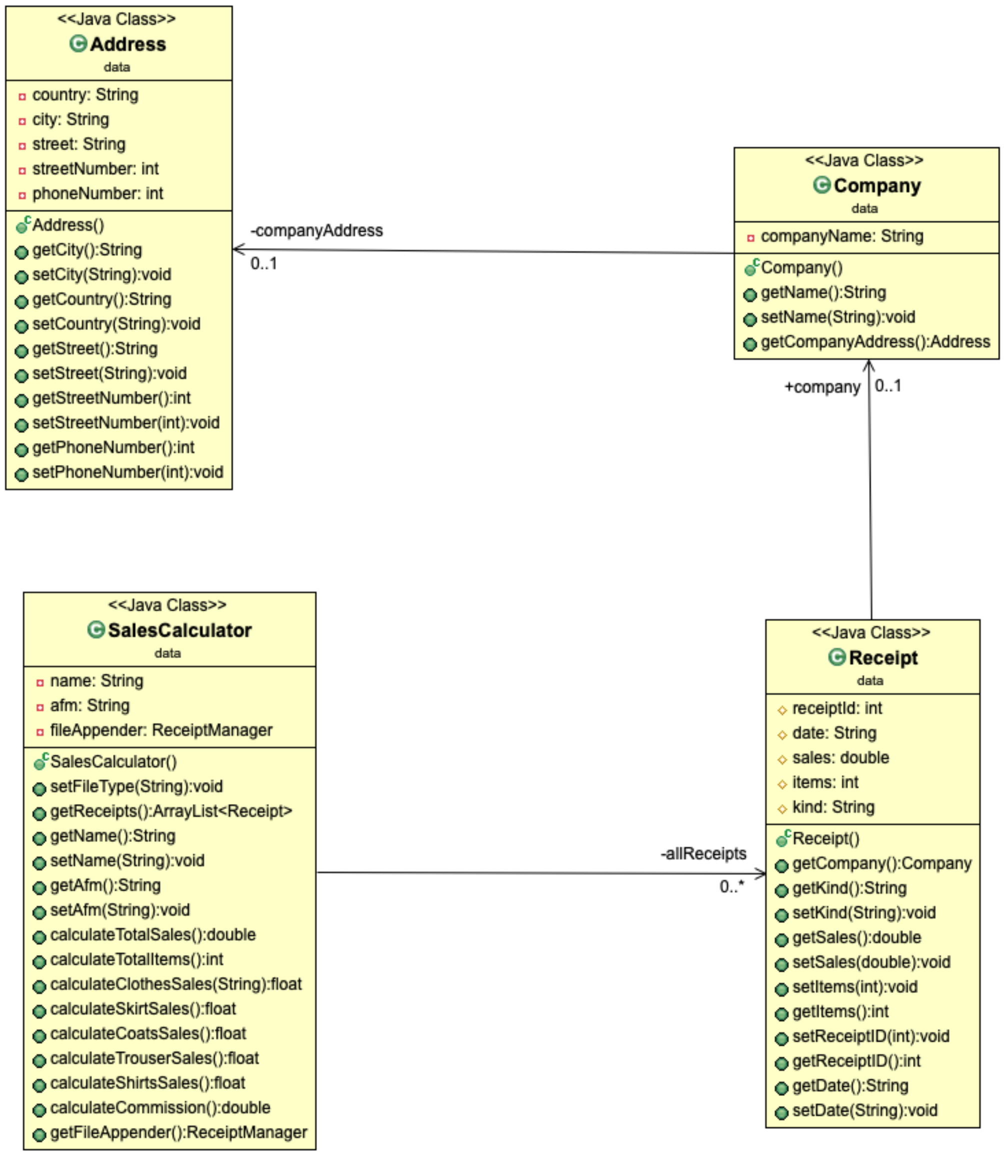
ARCHITECTURE

UML PACKAGE DIAGRAM



DETAILED DESIGN

• Package: data



<u>Problems we had to deal with:</u>

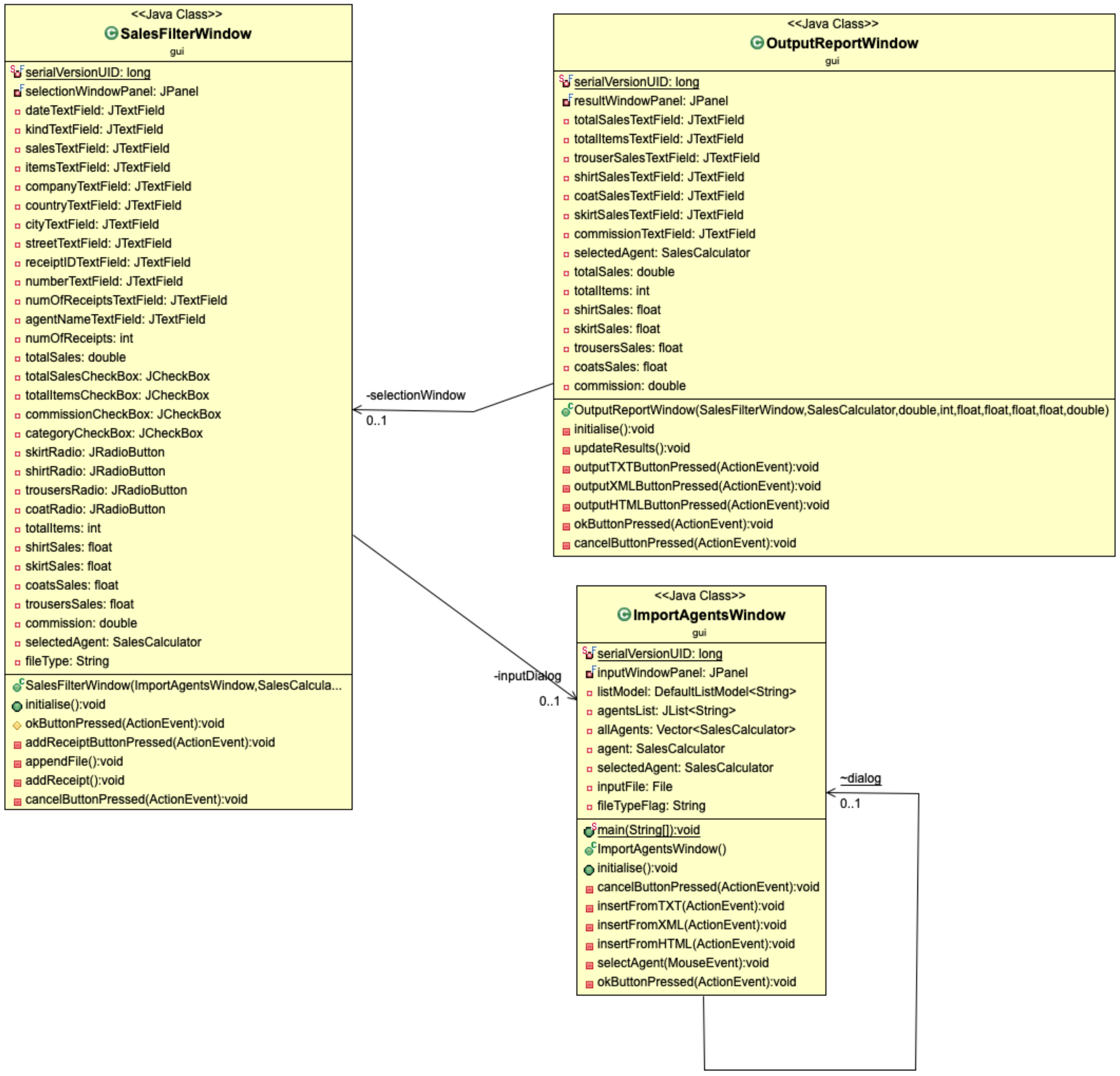
1. Lazy classes. We removed the classes Coat, Shirt, Shirt, Trouser because they didn't do anything important.

2. We changed the name of Agent class to SalesCalculator so as to reflect better to the class' role.

3. Duplicate code in SalesCalculator class. We removed the methods calculateSkirtSales(), calculateCoatsSales(), calculateTrouserSales(), calculateShirtsSales() because the code was the same except from the variables, so we replaced those methods with calculateClothesSales(String cloth).

4. In SalesCalculator class we replaced the JavaVector data structure with ArrayList.

• Package: gui



<<Java Class>>
SalesFilterWindow
gui
serialVersionUID: long
selectionWindowPanel: JPanel
dateTextField: JTextField
kindTextField: JTextField
salesTextField: JTextField
itemsTextField: JTextField
companyTextField: JTextField
countryTextField: JTextField
cityTextField: JTextField
streetTextField: JTextField
receiptIDTextField: JTextField
numberTextField: JTextField
numOfReceiptsTextField: JTextField
agentNameTextField: JTextField
numOfReceipts: int
totalSales: double
totalSalesCheckBox: JCheckBox
totalItemsCheckBox: JCheckBox
commissionCheckBox: JCheckBox
categoryCheckBox: JCheckBox
skirtRadio: JRadioButton
shirtRadio: JRadioButton
trousersRadio: JRadioButton
coatRadio: JRadioButton
totalItems: int
shirtSales: float
skirtSales: float
coatsSales: float
trousersSales: float
commission: double
selectedAgent: SalesCalculator
fileType: String
SalesFilterWindow(ImportAgentsWindow,SalesCalcula...
initialise():void
okButtonPressed(ActionEvent):void
addReceiptButtonPressed(ActionEvent):void
appendFile():void
addReceipt():void
cancelButtonPressed(ActionEvent):void
<<Java Class>>
OutputReportWindow
gui
serialVersionUID: long
resultWindowPanel: JPanel
totalSalesTextField: JTextField
totalItemsTextField: JTextField
trouserSalesTextField: JTextField
shirtSalesTextField: JTextField
coatSalesTextField: JTextField
skirtSalesTextField: JTextField
commisionTextField: JTextField
selectedAgent: SalesCalculator
totalSales: double
totalItems: int
shirtSales: float
skirtSales: float
trousersSales: float
coatsSales: float
commission: double
OutputReportWindow(SalesFilterWindow,SalesCalculator,double,int,float,float,float,float,double)
initialise():void
updateResults():void
outputTXTButtonPressed(ActionEvent):void
outputXMLButtonPressed(ActionEvent):void
outputHTMLButtonPressed(ActionEvent):void
okButtonPressed(ActionEvent):void
cancelButtonPressed(ActionEvent):void
-selectionWindow
0..1
-inputDialog
0..1
<<Java Class>>
ImportAgentsWindow
gui
serialVersionUID: long
inputWindowPanel: JPanel
listModel: DefaultListModel<String>
agentsList: JList<String>
allAgents: Vector<SalesCalculator>
agent: SalesCalculator
selectedAgent: SalesCalculator
inputFile: File
fileTypeFlag: String
main(String[]):void
ImportAgentsWindow()
initialise():void
cancelButtonPressed(ActionEvent):void
insertFromTXT(ActionEvent):void
insertFromXML(ActionEvent):void
insertFromHTML(ActionEvent):void
selectAgent(MouseEvent):void
okButtonPressed(ActionEvent):void
~dialog
0..1

<u>Problems we had to deal with:</u>

1. InputWindow, SelectionWindow, ResultWindow: We changed those classes' name to ImportAgentsWindow, SalesFilterWindow, OutputReportWindow accordingly.

2. In SalesFilterWindow the method addReceiptButtonPressed(ActionEvent evt) has a complex conditional statement witch we improved. In the legacy application the method was checking that if every field was empty so as to show an error. We improved that by adding those fields to a list and the only conditional statement now is if(checkList.isEmpty()).

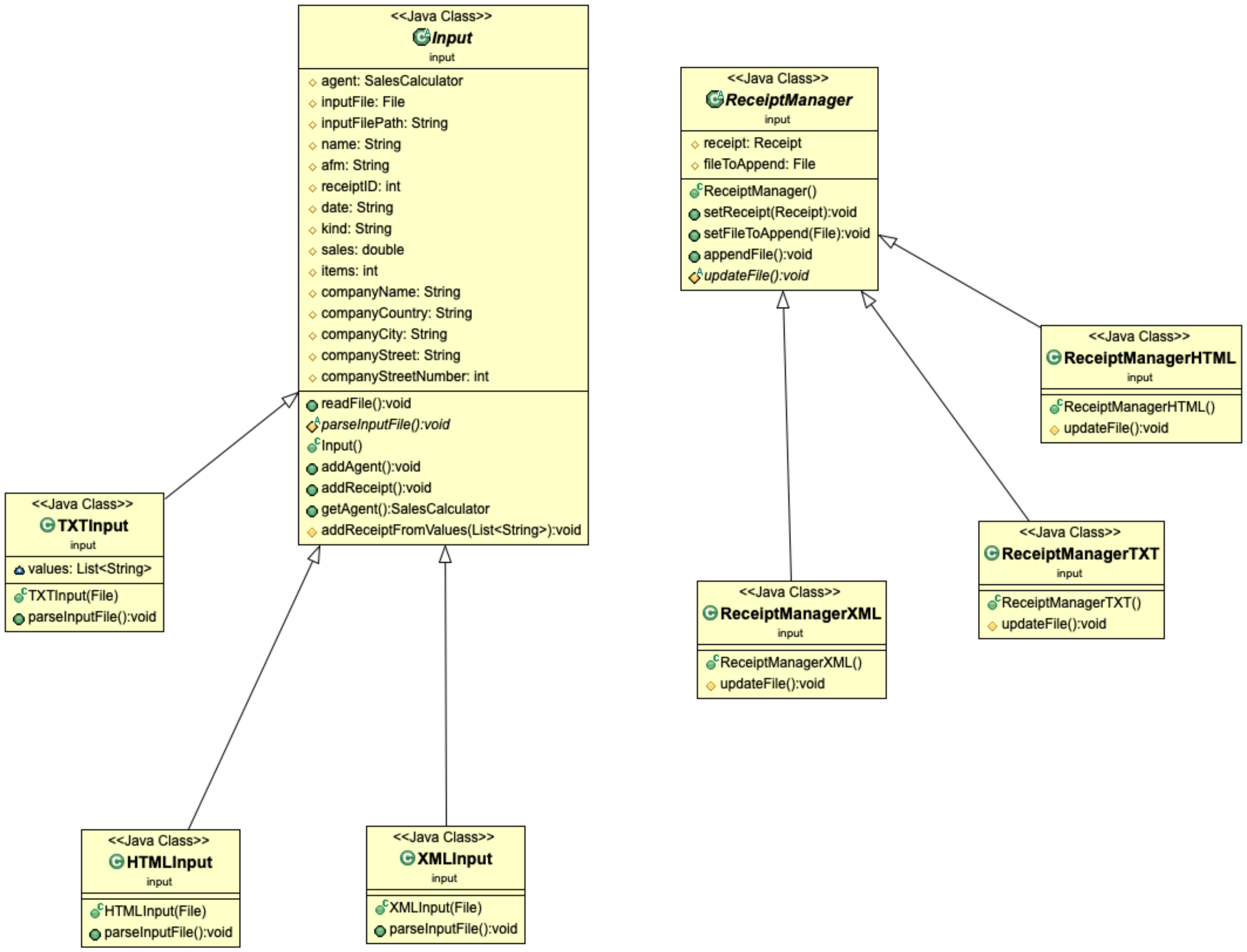
3. SalesFilterWindow.appendFile(): In order to decrease the chains of methods calls we created a receipt object.

4. SalesFilterWIndow.okButtonPressed(): We simplified the multiple if-else statements. We checked if shirt, skirts, trousers, coat buttons are selected only if the category checked box is selected.

<u>Extra:</u>

In OutputReportWindow we refactored the gui to allow the user to select a particular file for saving a report. In addition to this, the user can also, save a report to a new file with a name the user chooses. In the legacy application there was only one predetermined path to save a report, so we made the appropriate changes to the gui so as, when the user chooses to save a report, the application pops up a fileChooser window to select the path. Finally we refactored the writeToFIle(String filePath) method at output package so as to get the path from gui.

- **Package: input**



<u>Problems we had to deal with:</u>

1. ReceiptManager, ReceiptManagerTXT, ReceiptManagerXML: Firstly, we moved the classes from data package that they were in the legacy application since the classes perform IO operations. Secondly, their old names (FileAppender, FileAppenderTXT, FileAppenderXML) did not reflect on their functionality, so we gave them better names.

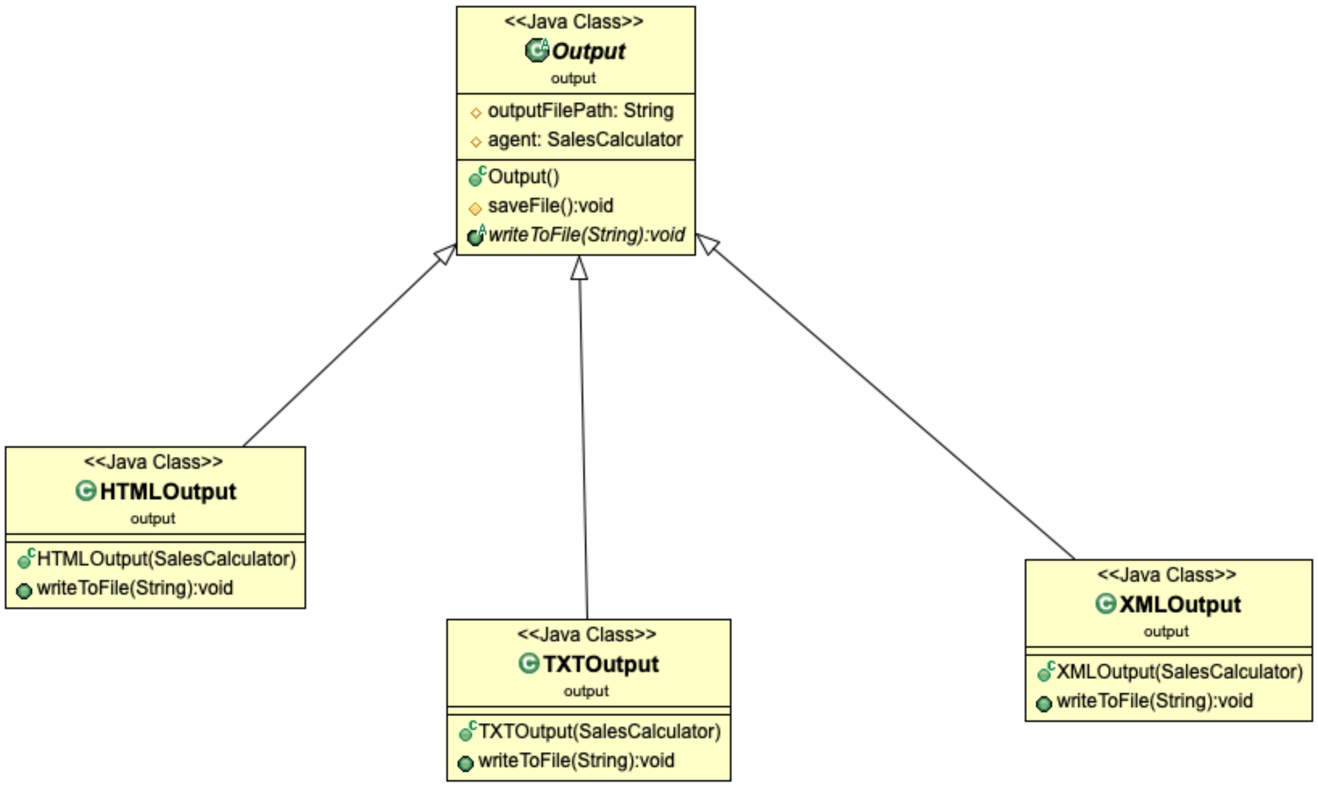
2. ReceiptManager: The problem is primitive obsession. The class in the legacy application had many fields of a primitive type, so we initialised a Receipt object that encapsulates the primitive data. We also created a setReceipt(Receipt receipt) method to get the data from the super class. Finally, we removed all unnecessary methods (setters-getters) as we can now get the data from the Receipt object.

3. ReceiptManager, ReceiptManagerTXT, ReceiptManagerXML, ReceiptManagerHTML: The children classes follow the same algorithmic steps in the updateFIle(). We extracted the common algorithm in a template method that we put in the ReceiptManger (appendFile()).

4. TXTInput: We replaced the algorithm to parse txt input files with a better and simpler one. Given that the order of the elements within the txt file is predetermined and fixed, we started parsing the file so as to find the name of the afm of the agent. Then, we iterate through the file to find the number of the receipts. After that, we iterate every receipt to find the values of the contextual receipt.

5. Input, TXTInput, XMLInput, HTMLInput: The children classes follow the same algorithmic steps in the parseInputFIle(). We extracted the common algorithm in a template method that we put in the Input (updateFIle()).

• **Package: output**



<u>Problems we had to deal with:</u>

1. Output, TXTOutput, XMLOutput: Their old names (Report, TXTReport, XMLReport) did not reflect on their functionality, so we gave them better names.

2. Output, TXTOutput, XMLOutput, HTMLOutput: The children classes follow the same algorithmic steps in the writeToFile(String fullPathName). We extracted the common algorithm in a template method that we put in the Output (saveFIle()).

EXTENSION TASK

We provided input to the application via an HTML file and we can also store report in an HTML file.

CLASSES RESPONSIBILITIES AND COLLABORATIONS (CRC CARDS)

• **Package data:**

Class Name: Address	
Responsibilities	**Collaborations**
▪ Address information(getters-setters)	-

Class Name: Company	
Responsibilities	**Collaborations**
▪ Company Information(getters-setters)	▪ Address

Class Name: Receipt	
Responsibilities	**Collaborations**
▪ Receipt information(getters-setters)	▪ Company

Class Name: SalesCalculator	
Responsibilities	**Collaborations**
▪ Information about the agent(getters-setters) ▪ Calculates sales, items and	▪ Receipt ▪ ReceiptManager

• **<u>Package gui:</u>**

Class Name: ImportAgentsWindow	
Responsibilities	**Collaborations**
▪ MainApp ▪ Imports file ▪ Selects Agent	▪ SalesCalculator ▪ TXTInput ▪ XMLInput ▪ HTMLInput

Class Name: SalesFilterWindow	
Responsibilities	**Collaborations**
▪ Allows user to select the preferred receipt's information ▪ Appends new receipt	▪ ImportAgentsWindow ▪ SalesCalculator ▪ Receipt

Class Name: ImportAgentsWindow	
Responsibilities	**Collaborations**
▪ Presents the selected information ▪ Allows user to store the report in the preferred file path	▪ SalesFilterWindow ▪ SalesCalculator ▪ TXTOutput ▪ XMLOutput ▪ HTMLOutput

• **Package input:**

Class Name: Input	
Responsibilities	**Collaborations**
▪ Abstract Class ▪ Provides the template method to children classes to parse the input file ▪ Stores agents' information ▪ Stores the receipt information	▪ SalesCalculator ▪ Receipt

Class Name: TXTInput	
Responsibilities	**Collaborations**
▪ Parses txt input file	▪ Input

Class Name: XMLInput	
Responsibilities	**Collaborations**
▪ Parses xml input file	▪ Input

Class Name: HTMLInput	
Responsibilities	**Collaborations**
▪ Parses html input file	▪ Input

Class Name: ReceiptManager	
Responsibilities	**Collaborations**
▪ Abstract Class ▪ Provides the template method to children classes to append the receipt information	▪ Receipt

Class Name: ReceiptManagerTXT	
Responsibilities	**Collaborations**
▪ Updates the txt file with new information	▪ ReceiptManager

Class Name: ReceiptManagerXML	
Responsibilities	**Collaborations**
▪ Updates the xml file with new information	▪ ReceiptManager

Class Name: ReceiptManagerHTML	
Responsibilities	**Collaborations**
▪ Updates the html file with new information	▪ ReceiptManager

- **Package output:**

Class Name: Output	
Responsibilities	**Collaborations**
▪ Abstract Class ▪ Provides the template method to children classes to store the report	▪ SalesCalculator

Class Name: TXTOutput	
Responsibilities	**Collaborations**
▪ Writes the txt report	▪ Output

Class Name: XMLOutput	
Responsibilities	**Collaborations**
▪ Writes the xml report	▪ Output

Class Name: HTMLOutput	
Responsibilities	**Collaborations**
▪ Writes the html report	▪ Output

TESTS

- <u>TXTInputTest, XMLInputTest, HTMLInputTest:</u> As a setup to each test we called a helper method that we implemented which we create a new .txt/.xml/.html file respectively. Then, we initialise a TXTInput/XMLInput/HTMLInput object and we parse the created test file. Finally, we check if every value of the receipt we created by calling the correct methods using the TXTInput/XMLInput/ HTMLInput object is the same as the actual value of the file.

- <u>TXTOutputTest, XMLOutputTest, HTMLOutputTest:</u> We give the previously created file (.txt/.xml/.html) as an input and we create a TXTOutput/XMLOutput/HTMLOutput object and we write the report to a new file. Finally, we calculated the actual values of the report and we check if the values resulting from calling the correct methods using our object are the same.
- A Test Suite (AllTests.java) was used to run all tests at once.