SALES COMMISSIONS APPLICATION

OVERALL REPORT

VERSION <1.0>

<Fill here the group members, Name and AM>

TABLE OF CONTENTS

Introduction 4

Refactored Design 4

Use Cases 4

Architecture 7

Detailed Design 7

Classes Responcibilities and Collaborations (CRC CARDS) 12

INTRODUCTION

The general purpose of the application is to manage the sales made and the commissions attributed to the sales representatives of a small clothing company. The objective of the report is to provide a refactored design of the sales commissions application.

REFACTORED DESIGN

USE CASES

Specify the use cases of the application.

Use case ID	UC1: LoadReceipt
Actors	User
Pre conditions	-
Main flow of events	1. The use case starts when the user selects <<Εισαγωγή από TXT/XML>> from the menu. 2. The system opens a window where user can select the txt and xml file. 3. The user selects his/her wanted receipt file. 4. The system loads the corresponding representative.
Alternative flow 1	If the user selects a file format different than txt or xml, no receipt file is loaded, and a warning window pops up.
Post conditions	The representative's info is loaded in the system.

Use case ID	UC2: SelectRepresentative
Actors	User
Pre conditions	The user must select a representative.
Main flow of events	1. The use case starts when the user has selected a representative from the representatives' list and clicks the OK button. 2. The system opens a new window about the selected representative.
Alternative flow 1	-
Post conditions	The current window is about the selected representative.

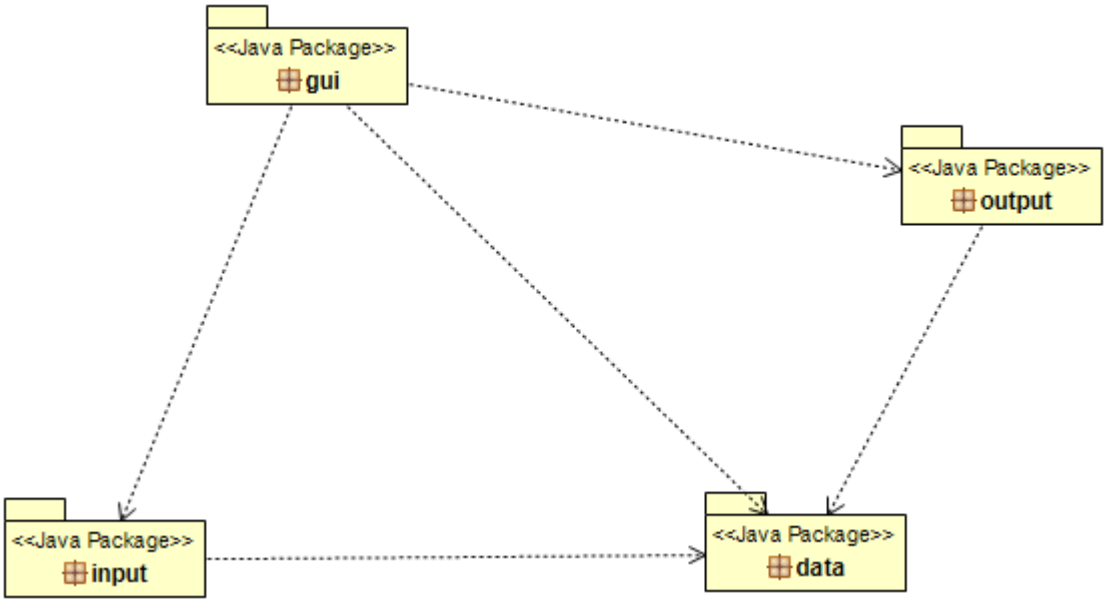
Use case ID	UC3: ShowGeneralInfo
Actors	User
Pre conditions	The user must select the box about information he wants to access.
Main flow of events	1. The use case starts when the user has checked in a box about the desired information and clicks the OK button. 2. The system opens a new window where user can watch the information that he chose.
Alternative flow 1	-
Post conditions	The system provides a new menu for the user.

Use case ID	UC4: SaveGeneralReport
Actors	User
Pre conditions	The system must be in show info state of UC3.
Main flow of events	1. The use case starts when the user selects <<Εξαγωγή TXT >> or <<Εξαγωγή XML >> from the menu. 2. The system saves the general report in txt or xml format.
Alternative flow 1	-
Post conditions	The report is saved in the system.

Use case ID	UC5: AddNewReceipt
Actors	User
Pre conditions	A representative must be selected.
Main flow of events	1. The use case starts when the user selects <<Προσθήκη νέας απόδειξης>> from the menu. 2. The system opens a form where user can input the receipt's info. 3. The user fills the form with the receipt's info and then clicks <<Προσθήκη>>. 4. The system adds this new receipt to the receipt file.
Alternative flow 1	If the user leaves a form's input box empty the system pops a warning window.
Post conditions	The new receipt is added.

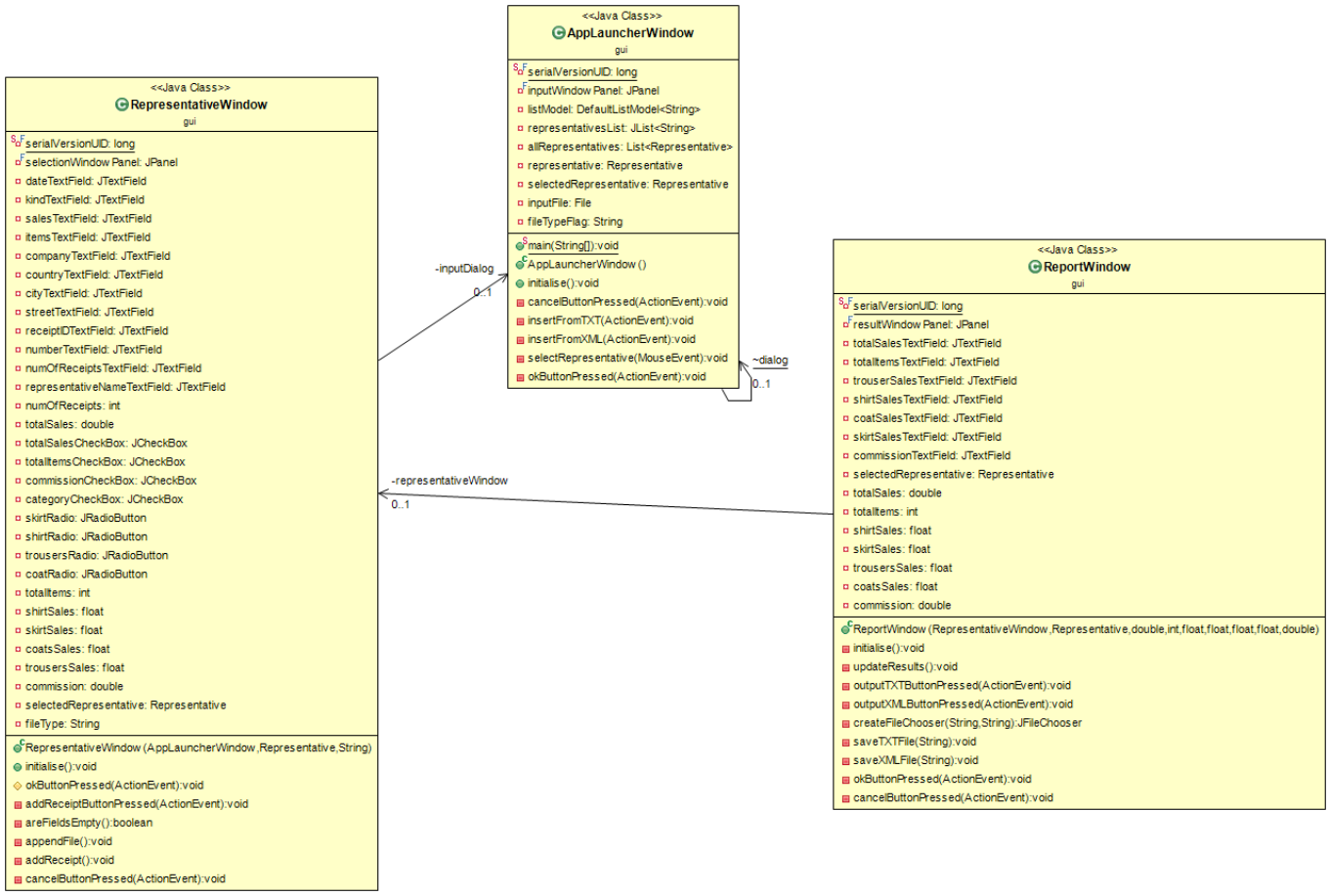
ARCHITECTURE

package diagram

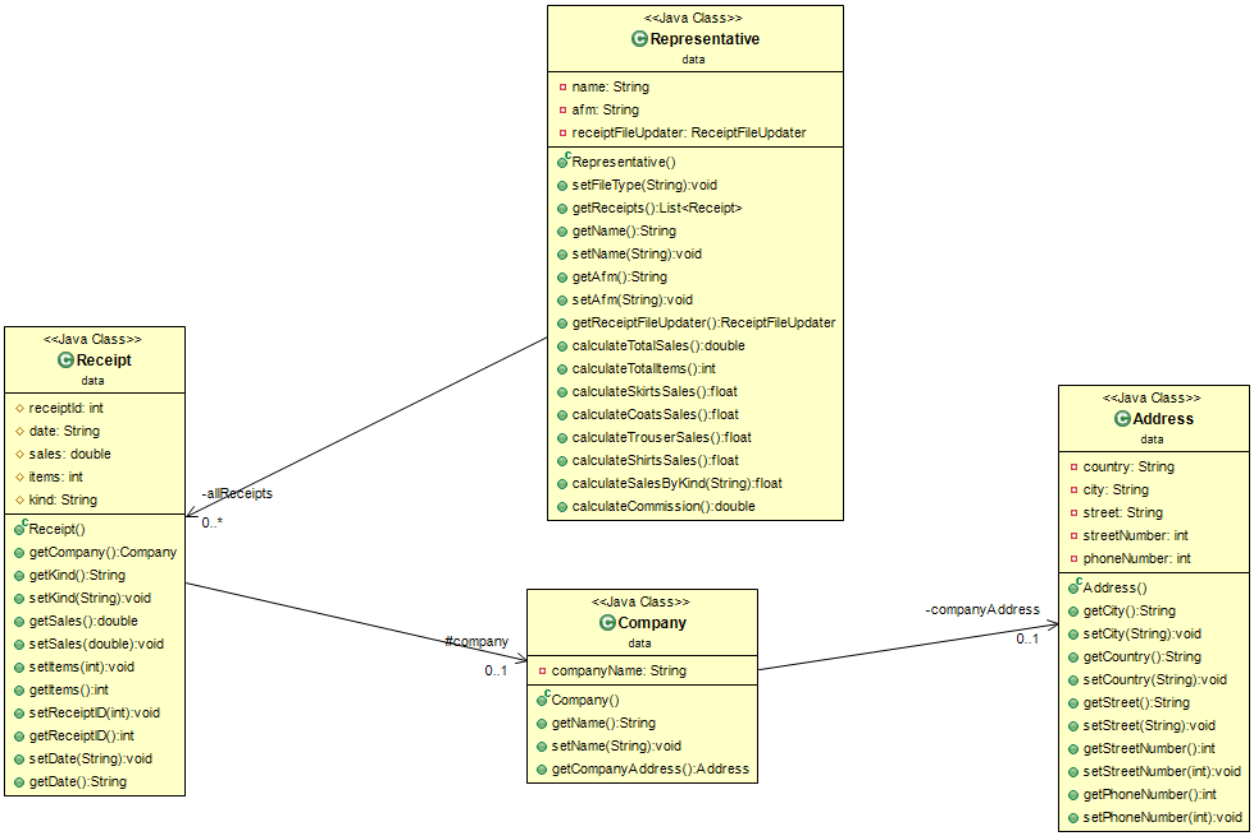


DETAILED DESIGN

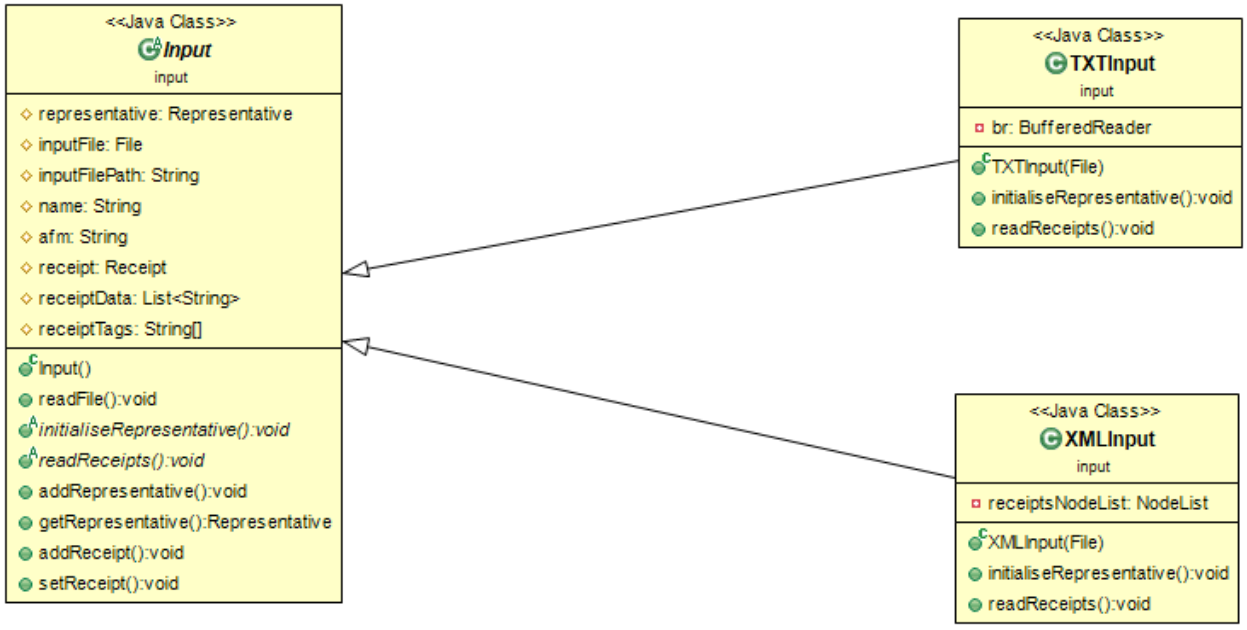
gui package


<<Java Class>>
RepresentativeWindow
gui
serialVersionUID: long
selectionWindow Panel: JPanel
dateTextField: JTextField
kindTextField: JTextField
salesTextField: JTextField
itemsTextField: JTextField
companyTextField: JTextField
countryTextField: JTextField
cityTextField: JTextField
streetTextField: JTextField
receiptIDTextField: JTextField
numberTextField: JTextField
numOfReceiptsTextField: JTextField
representativeNameTextField: 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
selectedRepresentative: Representative
fileType: String
RepresentativeWindow (AppLauncherWindow,Representative,String)
initialise():void
okButtonPressed(ActionEvent):void
addReceiptButtonPressed(ActionEvent):void
areFieldsEmpty():boolean
appendFile():void
addReceipt():void
cancelButtonPressed(ActionEvent):void
<<Java Class>>
AppLauncherWindow
gui
serialVersionUID: long
inputWindow Panel: JPanel
listModel: DefaultListModel<String>
representativesList: JList<String>
allRepresentatives: List<Representative>
representative: Representative
selectedRepresentative: Representative
inputFile: File
fileTypeFlag: String
main(String[]):void
AppLauncherWindow ()
initialise():void
cancelButtonPressed(ActionEvent):void
insertFromTXT(ActionEvent):void
insertFromXML(ActionEvent):void
selectRepresentative(MouseEvent):void
okButtonPressed(ActionEvent):void
-inputDialog
0..1
~dialog
0..1
<<Java Class>>
ReportWindow
gui
serialVersionUID: long
resultWindow Panel: JPanel
totalSalesTextField: JTextField
totalItemsTextField: JTextField
trouserSalesTextField: JTextField
shirtSalesTextField: JTextField
coatSalesTextField: JTextField
skirtSalesTextField: JTextField
commissionTextField: JTextField
selectedRepresentative: Representative
totalSales: double
totalItems: int
shirtSales: float
skirtSales: float
trousersSales: float
coatsSales: float
commission: double
ReportWindow (RepresentativeWindow,Representative,double,int,float,float,float,float,double)
initialise():void
updateResults():void
outputTXTButtonPressed(ActionEvent):void
outputXMLButtonPressed(ActionEvent):void
createFileChooser(String,String):JFileChooser
saveTXTFile(String):void
saveXMLFile(String):void
okButtonPressed(ActionEvent):void
cancelButtonPressed(ActionEvent):void
-representativeWindow
0..1

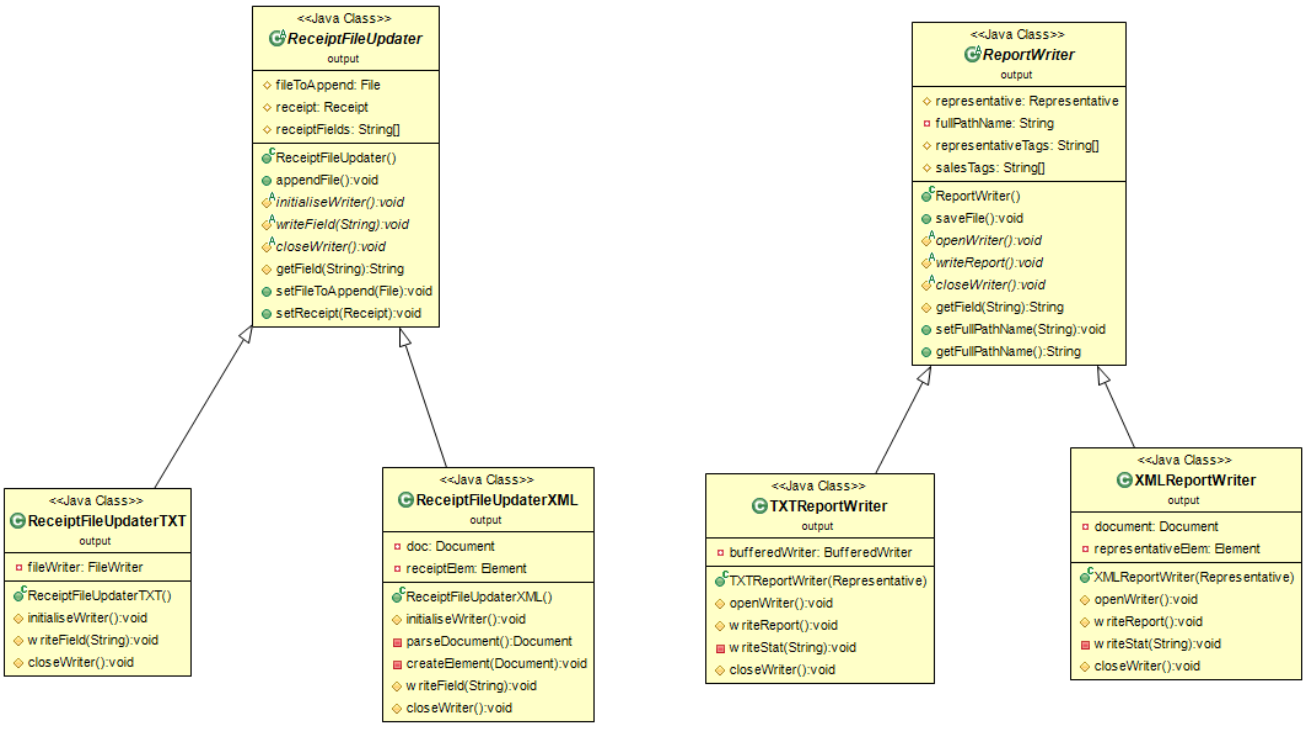
data package



input package



output package



Addressing the different **problems** of the old design

In the data package the old design was using the classes Coat, Shirt, Skirt, Trouser for the different kinds of a receipt. These classes were just used to assign the kind of a receipt, so they were removed, and the kind of receipt is now assigned by the Receipt class.

The Agent class was renamed to **Representative** to reflect the role of the class better. The methods to calculate sales by kind had duplicate code in each method and so the common part for the calculation was moved in the new calculateSalesByKind() method. CalculateCommission() method was refactored to a simpler algorithmic logic. Also, deprecated Vector data structure that was holding the receipts was replaced by a List.

The **FileAppender** abstract class and its subclasses were renamed to **ReceiptFileUpdater** and were placed in output package to better correspond their goal. The appendFile() method was moved to the ReceiptFileUpdater class because the algorithmic steps were the same and new abstract methods were created to fulfill the different implementations in the concrete classes ReceiptFileUpdaterTXT and ReceiptFileUpdaterXML.

Similarly, in the Input package the common algorithm of readFile() method in the **TXTInput** and **XMLInput** classes was extracted as a template method in the Input class and the different parts are now implemented by the abstract methods initialiseRepresentative() and readReceipts().

Report classes were renamed to more representative names **ReportWriter**, ReportWriterTXT, ReportWriterXML and the base algorithm of saveFile() method was put in the base class Report as before.

Finally, in the **gui** package better names were given to the classes to better reflect their responsibilities. The user can now save the report in a particular file instead of using a predefined path. The complex conditional statement of addReceiptButtonPressed() was extracted in a new boolean method and the okButtonPressed() method's multiple if-else statements were simplified.

CLASSES RESPONSIBILITIES AND COLLABORATIONS (CRC CARDS)

Class Name: Address	
Responsibilities	**Collaborations**
This class is responsible to keep a company's address information.	Company

Class Name: Company	
Responsibilities	**Collaborations**
This class is responsible to keep a company's information.	Address

Class Name: Receipt	
Responsibilities	**Collaborations**
This class represents a receipt and is responsible to keep receipt's information.	Company

Class Name: Representative

Responsibilities	**Collaborations**
This class represents a representative and is responsible to keep track of his info and receipts.	ReceiptFileUpdater Receipt

Class Name: Input	
Responsibilities	**Collaborations**
This class is responsible for reading the representative's file with the receipts.	-

Class Name: TXTInput	
Responsibilities	**Collaborations**
This class is responsible to read the receipts from a txt file.	Input

Class Name: XMLInput	
Responsibilities	**Collaborations**
This class is responsible to read the receipts from a xml file.	Input

Class Name: ReceiptFileUpdater	
Responsibilities	**Collaborations**
This class is responsible to add a new receipt in the representative's file.	Receipt

Class Name: ReceiptFileUpdaterTXT	
Responsibilities	**Collaborations**
This class is responsible to add a new receipt in a txt file.	ReceiptFileUpdater Receipt

Class Name: ReceiptFileUpdaterXML	
Responsibilities	**Collaborations**
This class is responsible to add a new receipt in a xml file.	ReceiptFileUpdater Receipt

Class Name: ReportWriter

Responsibilities	**Collaborations**
This class is responsible to produce general reports.	Representative

Class Name: TXTReportWriter	
Responsibilities	**Collaborations**
This class produces reports in a txt file format.	ReportWriter Representative

Class Name: XMLReportWriter	
Responsibilities	**Collaborations**
This class produces reports in a xml file format.	ReportWriter Representative

Class Name: AppLauncherWindow	
Responsibilities	**Collaborations**
This class provides the starting UI for inserting	Representative

representatives to the system.	

Class Name: Representative	
Responsibilities	**Collaborations**
This class is responsible for the graphical interface to access a representative's info and add a new receipt.	Representative

Class Name: ReportWindow	
Responsibilities	**Collaborations**
This class provides the graphical interface to save a file in a user's wanted location.	Representative