INCOME TAX CALCULATOR 2022

OVERALL REPORT

VERSION 1.0

<Αθανάσιος Κουρέας, 4392>

<Αθανάσιος Παπαποστόλου, 4147>

INTRODUCTION

Σκοπός της εργασίας είναι το refactoring ενος codebase για τη διαχείρηση φορολογικών δεδομένων για τους πολίτες της Minnesota. Η αρχική εφαρμογή είχε 3 βασικά υποσυστήματα σχετικά με τη **γραφική διεπαφή**, την **είσοδο/έξοδο**, και τη **διαχείριση δεδομένων**. Σαν πρώτη εντύπωση παρατηρήσαμε πολλαπλά code smells σε όλο το φάσμα της εφαρμογής. Η γραφική διεπαφή διαχειρίζονταν μερικές business logic λειτουργίες, το TaxpayerManager είχε αυξημένη πολυπλοκότητα και έκρυβε domain specific αντικείμενα. Το Taxpayer είχε conditionals και iterations που έπρεπε να αφαιρεθούν, ενώ τα βοηθητικά I/O αντικείμενα είχαν duplication και υλοποιούσαν την κληρονομικότητα λανθασμένα.

Στην προσπάθεια για ανακατασκευή της εφαρμογής, ακολουθήσαμε **use-case-driven** προσέγγιση. Συνοπτικά εκφράσαμε το use-case model με **requirements model**, το δομήσαμε με **analysis model**, το σχεδιάσαμε με το **design model**, υλοποιήσαμε με **implementation model**, και το ελέγξαμε με το **testing model**. Παρακάτω αναλύουμε κάθε μέρος της υλοποίησης με βάση τα models.

REQUIREMENTS MODEL

Διαβάζοντας τα requirements της εφαρμογής στο requirements model εξηγούμε βασικές σχεδιαστικές απαιτήσεις, την επικοινωνία μεταξύ των actors, βασικά δεδομένα και use cases για τις λειτουργίες της εφαρμογής και την αλληλεπίδραση με τους actors.

ACTORS

Το σύστημα έχει **έναν actor**, τον χρήστη της εφαρμογής, και παρόλο που οι απαιτήσεις θεωρητικά επιτρέπει πολλούς χρήστες να αλληλεπιδράσουν με πολλά instances της εφαρμογής, οι actors δεν επικοινωνούν μεταξύ τους. Επομένως προυποθέτουμε σχετικά απλή υλοποίηση όπου ο χρήστης επεξεργάζεται δεδομένα για φορολογουμένους και τις αποδείξεις που καταθέτουν μέσω συγκεκριμένων λειτουργιών της εφαρμογής.

DATA REQUIREMENTS

Η εφαρμογή διαχειρίζεται δεδομένα για σύνολο φορολογουμένων. Κάθε φορολογούμενος έχει **ονοματεπώνυμο**, **ΑΦΜ**, **οικογενειακή κατάσταση**, **εισόδημα** και ένα σύνολο από **αποδείξεις** που έχει συλλέξει. Οι αποδείξεις ανήκουν σε μία από τις εξής κατηγορίες: **Entertainment**, **Basic**, **Travel**, **Health**, **Other**. Οι αποδείξεις χαρακτηρίζονται από **κωδικό** απόδειξης, **ημερομηνία** έκδοσης, **κατηγορία**, **ποσό**, και **επιχείρηση** που εκδόθηκε. Οι επιχειρήσεις χαρακτηρίζονται από **όνομα** και **διεύθυνση**. Οι διευθύνσεις χαρακτηρίζονται από **χώρα**, **πόλη**, **οδό** και **αριθμό**. Η εφαρμογή επεξεργάζεται τα εξής δεδομένα με τα use cases που παρουσιάζουμε παρακάτω.

USE CASES

Use case ID	***UC1 - Load Taxpayer Information from File***
Data	Taxpayer TRN, File Type
Main flow of events	1. The use case starts when the user of the system issues `Load Taxpayer` command. 2. User is prompted to give taxpayer's registration number as well a filetype for loading. 3. System uses TRN and Type provided to find the corresponding file. 4. System loads all taxpayer information along with receipt data for the current taxpayer. 5. System stores data in memory.
Alternative flow 1	Invalid TRN: 1. User provided a tax registration number that is invalid as a format. 2. User is notified with: `The tax registration number must have only digits.`. 3. UC1 is executed again.
Alternative flow 2	Non Existent TRN: 1. User provided a valid tax registration number however the appropriate file does not exist. 2. User is notified with: `This file does not exist.`. 3. UC1 is executed again.

Alternative flow 3	Wrong File Type: 1. User provided a file type that is invalid or not handled by the system. 2. User is notified with: `Please check your file format and try again.`. 3. UC1 is executed again.
Alternative flow 4	Wrong Taxpayer Status: 1. User provided a valid taxpayer however their status is invalid (not one of the types provided in requirements). 2. User is notified with: `Please check taxpayer's status and try again.`. 3. UC1 is executed again.
Alternative flow 5	Wrong Receipt Kind: 1. User provided a valid taxpayer however one of their receipts is of invalid kind (not one of the types provided in requirements). 2. User is notified with: `Please check receipts kind and try again.`. 3. UC1 is executed again.
Alternative flow 6	Wrong Receipt Date: 1. User provided a valid taxpayer however one of their receipts has an invalid date. 2. User is notified with: `Please make sure your date is DD/MM/YYYY and try again.`. 3. UC1 is executed again.

Use case ID	***UC2 - Select and Display Taxpayer Information***
Data	Taxpayer TRN
Main flow of events	1. The use case starts when the user of the system issues `Select Taxpayer` command. 2. User is prompted to provide the tax registration number of the user he wants to select. 3. System uses provided TRN to search in the list of taxpayers already loaded. 4. System displays taxpayer data about personal information and the list of his receipts.
Alternative flow 1	<u>Non Existent TRN</u>: 1. User has provided a TRN that does not exist in the list loaded in the system. 2. User is notified with: `This tax registration number isn' loaded.`. 3. Use case ends.
Alternative flow 2	<u>Invalid TRN</u>: 1. User provided an invalid format for a TRN. 2. User is notified with: `You must give a tax registration number.`. 3. Use case ends.
Alternative flow 3	<u>No Taxpayer Loaded</u>: 1. User tried to select a taxpayer from a list that has none loaded. 2. User is notified with: `There isn't any taxpayer loaded. Please load one first.`. 3. Use case ends.

Use case ID	***UC3 - Add Receipt***
Data	Receipt ID, Date, Kind, Amount, Company Name, Country, City, Street, Number, TRN
Main flow of events	1. The use case starts when the user of the system issues `Add Receipt` command. 2. User is prompted to provide information about the new receipt he wants to input into the system. 3. User provides a unique id, a date, kind and amount of the receipt, as well as the address of the company that issued the receipt. 4. System uses information provided to issue a new receipt. 5. System saves receipt in memory under the list of receipt of the current user (using provided TRN). 6. System executes UC5.
Alternative flow 1	<u>Existing Receipt ID</u>: 1. User has provided a receipt id that is not unique and already exists in the taxpayers list. 2. User is notified with: `Receipt ID already exists.`. 3. Use case ends.
Alternative flow 2	<u>Invalid Date</u>: 1. User has provided a receipt with a date that is not valid (as in DD/MM/YYYY). 2. User is notified with: `Please make sure your date is DD/MM/YYYY and try again`. 3. Use case ends.
Alternative flow 3	<u>Invalid Kind</u>: 1. User has provided a receipt with an invalid kind (not one of the types provided in requirements). 2. User is notified with: `Please check receipt kinds and try again`. 3. Use case ends.

Use case ID	***UC4 - Delete Receipt***
Data	Receipt ID, Taxpayer TRN
Main flow of events	1. The use case starts when the user of the system issues `Delete Receipt` command. 2. User is prompted to provide a receipt id to delete. 3. System deletes receipt with corresponding id from memory of the current taxpayer (using provided TRN). 4. System executes UC5.
Alternative flow 1	Invalid Receipt ID: 1. User has provided a receipt id that is invalid or one that does not exist in the list. 2. Use case ends.

Use case ID	***UC5 - Update Taxpayer Information in Files***
Data	Taxpayer TRN
Main flow of events	1. System opens files of all supported types using given current TRN for write. 2. System updates information on newly added or deleted receipts.

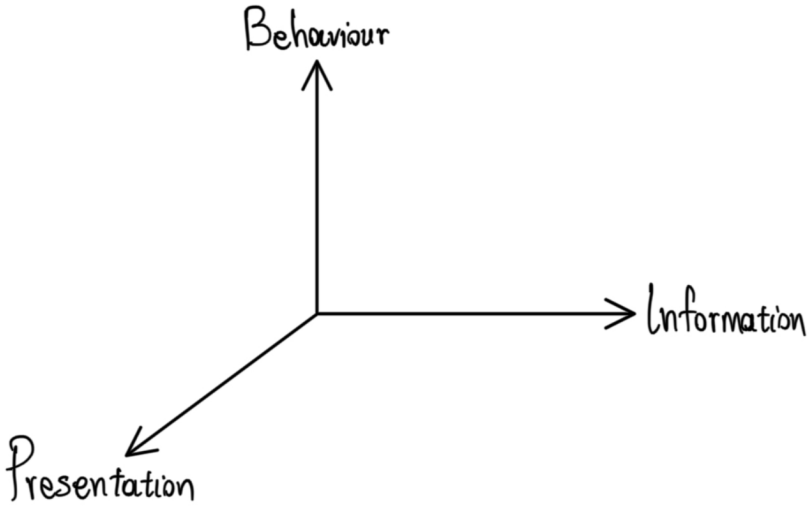
Use case ID	***UC6 - View Report***
Data	Taxpayer TRN
Main flow of events	1. Use case starts when the user of the system issues `View Report` command. 2. System aggregates tax information of current taxpayer (using provided TRN). 3. System calculates basic and total tax numbers, as well as the variation increase or decrease on said tax. 4. System gets information about the receipt amounts according to receipt types. 5. System displays a pie chart and bar chart with current expenditure types and tax analytics.

Use case ID	***UC7 - Save Data***
Data	Taxpayer TRN, File Type
Main flow of events	1. The use case starts when the user of the system issues `Save Data` command. 2. User selects the type of format to export to. 3. System uses TRN provided to create the corresponding log file with given export type. 4. System saves data in log file.
Alternative flow 1	Wrong File Format: 1. User has provided a file type that is not supported. 2. User is notified with: `Wrong file format.` 3. Use case ends.
Alternative flow 2	Wrong File Format: 1. Occurs when a filesystem error happens mid opening log file for write. 2. Use case ends.

Use case ID	***UC8 - Delete Taxpayer***
Data	Taxpayer TRN
Main flow of events	1. The use case starts when the user of the system issues `Delete Taxpayer` command. 2. User is prompted for a taxpayer TRN for deletion. 3. System deletes taxpayer with corresponding TRN from memory.
Alternative flow 1	No Taxpayer Loaded: 1. User tries to remove a taxpayer however no taxpayer is loaded yet. 2. User is notified with: `There isn't any taxpayer loaded. Please load one first.`. 3. Use case ends.
Alternative flow 2	Invalid Taxpayer TRN: 1. User has provided a TRN that is not valid (per requirements of the system). 2. User is notified with `Tax registration number is not a valid number`. 3. Use case ends.

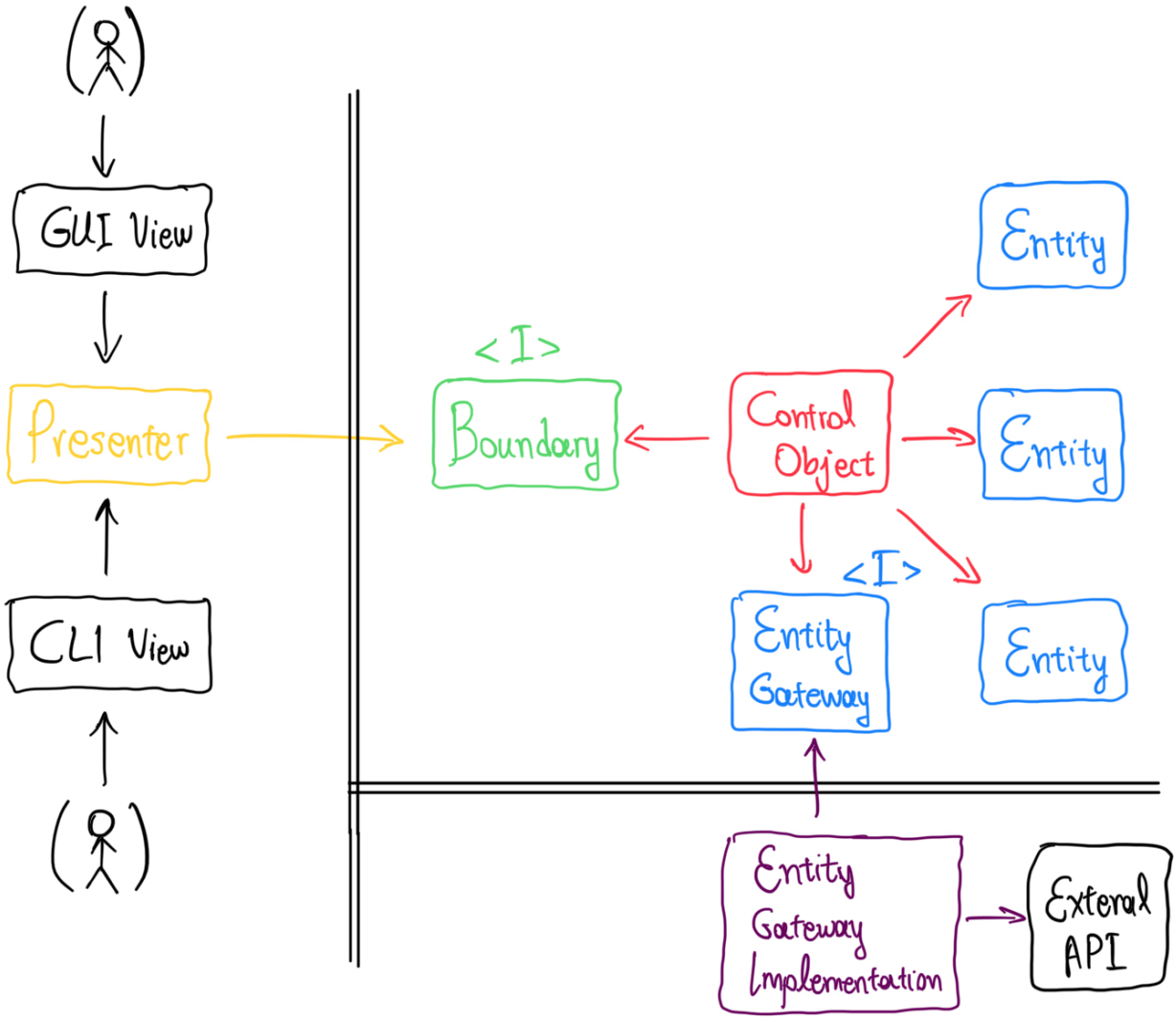
ANALYSIS MODEL

Σε αυτή τη φάση παρουσιάζουμε τα είδη αντικειμένων που απαιτούνται για σωστή υλοποίηση. Απαραίτητος είναι ο διαχωρισμός των concerns με τεχνική τέτοια έτσι ώστε με πρόσθεση ή αφαίρεση καινούριων use case, να διατηρείται το Single Responsibility Principle, και ιδιαίτερα το **Open Closed**. Υποθέτουμε ότι τα use cases μεταξύ τους δεν έχουν state dependencies. Πρέπει τα business logic, view, και data objects να επεκτείνονται με διαφορετική "κατεύθυνση" όσων αφορά τα dependencies. Σε αυτό το πλαίσιο θα χρησιμοποιήσουμε την **Entity-Interface-Control** (ή BCE ή BIE) αρχιτεκτονική.



BCE ARCHITECTURE

Το σύστημα θα αποτελείται από αντικείμενα δεδομένων **Entities**. Τα entities χρησιμοποιούνται και ενημερώνονται από τα **Control Objects**, τα οποία θα υλοποιούν τα use cases. Στον client έχουμε **Presenter** αντικείμενα τα οποία μεταφράζουν τα δεδομένα που παίρνουν από την εφαρμογή και τα μορφοποιούν ανάλογα για κάθε διεπαφή με τον χρήστη. Το front end και back end επικοινωνούν μέσω των **Boundary** αντικειμένων. Στην περίπτωση της Java αυτά θεωρούνται ως Java Interfaces. Όταν θέλουμε να συνδέσουμε εξωτερικές εφαρμογές όπως dbms, χρησιμοποιούμε το **Gateway Pattern**. Ένα entity χρησιμοποιείται σαν interface, και το gateway implementation χρησιμοποιείται σαν γέφυρα μεταξύ του external API και της εφαρμογής. Με αυτή τη σχεδίαση έχουμε ολοκληρωμένο διαχωρισμό μεταξύ της εφαρμογής, του client, και των εξωτερικών επεκτάσεων.



Σαν παράδειγμα, για το **UC1 - Load Taxpayer** θα κατασκευάζαμε ένα Boundary που οδηγεί την σύνδεση client-app, θα υλοποιούσαμε το Boundary με ένα Control Object που περιέχει τον ουσιαστικό κώδικα εκτέλεσης. Με την φόρτωση καινούριου Taxpayer θα αποθηκεύαμε τα δεδομένα σε αντίστοιχο Entity αντικείμενο. Θα κατασκευάζαμε Presenter αντικείμενο που χρησιμοποιεί τις δομές που παίρνει από το Boundary, και στη συνέχεια το αντοίστοιχο View αντικείμενο που χρησιμοποιεί τα δεδομένα και τα προβάλει στην διεπαφή. Ως τρόπο αποθήκευσης πολλών Taxpayer θα είχαμε ένα εξωτερικό DMBS το οποίο εκθέτει δημόσιο API, και μέσω Gateway θα το συνδέαμε σαν Entity στην εφαρμογή.

DESIGN MODEL

Ακολουθώντας το analysis model, μεταφράζουμε την ανάλυση σε κώδικα. Συνοποτικά έχουμε Entities για **Taxpayers** και **Receipts**. Σαν αντικείμενα διαχειρίζονται πληροφορίες και μπορούν να αλλάξουν το εσωτερικό τους state. Βοηθητικά έχουμε αντικείμενα **Date**, **Company**, **Address** για πιο αναλυτική δομή, και για διαφορετικά είδη Taxpayers χρησιμοποιούμε κληρονομικότητα εξειδικεύοντας με βάση την οικογενειακή κατάσταση και εισόδημα.

Για κάθε use case έχουμε από ένα Boundary και όμοια έχουμε από ενα Control Object. Σαν σχεδίαση, λόγω της μονοδιάστατης επικοινωνίας με έναν actor, θα μπορούσαμε να χρησιμοποιήσουμε Command Pattern και ένα γενικό interface με πιθανό όνομα **UseCaseInterface**. Για κάθε use case επίσης έχουμε ζευγάρι από Presenter και View. Στην περίπτωση της εφαρμογής μας χρησιμοποιούμε Java Swing. Τα πακέτα της εφαρμογής θα έχουν ονόματα αντίστοιχα με τα ονόματα των use cases για ευδιάκριτο διαχωρισμό.

Σε αυτό το σημείο μπορούμε να σχεδιάσουμε τα πρώτα acceptance tests. Για κάθε use case μπορούμε να έχουμε 1-2 acceptance tests που επιβεβαιώνουν την επικοινωνία μεταξύ των components της αρχιτεκτονικής (όχι integration tests).

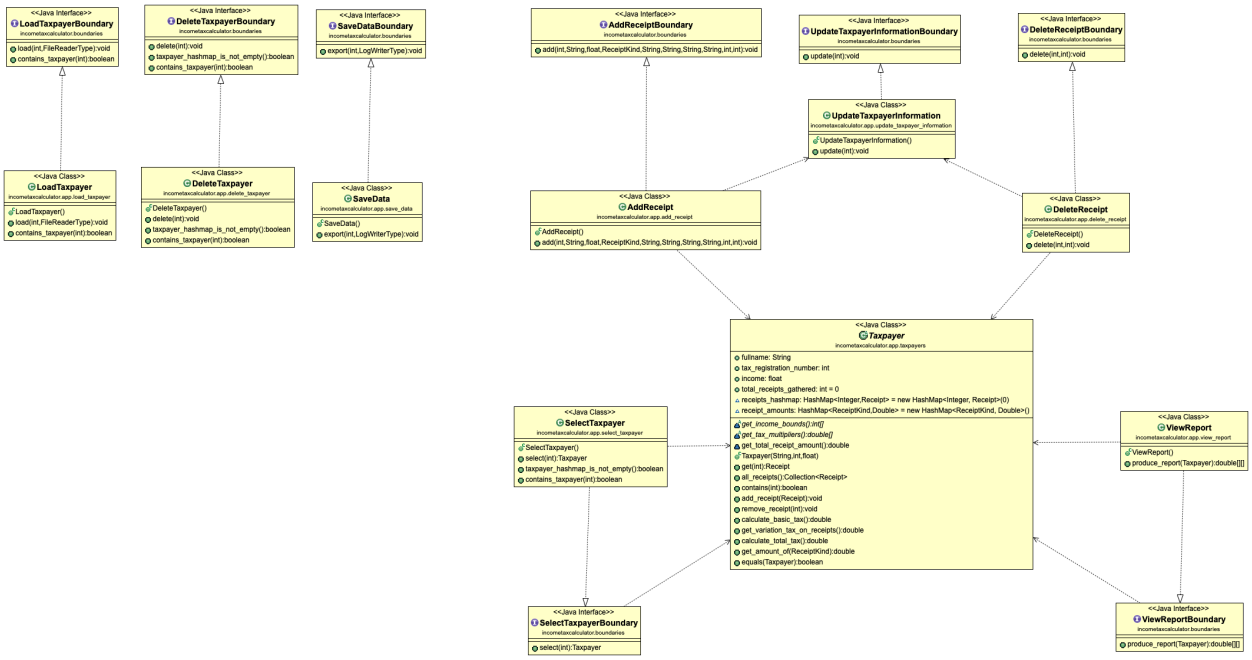
IMPLEMENTATION MODEL

Στο implementation model, υλοποιούμε κάθε πακέτο όσο γίνεται ξεχωριστά από την υπόλοιπη εφαρμογή. Συγκεκριμένα για την άσκηση έχουμε ήδη τον κώδικα. Ιδανικά η υλοποίηση θα είχε πραγματοποιηθεί με Behaviour ή Test Driven Development, ωστόσο εδώ δεν έχουμε unit test. Για το refactoring είναι απαραίτητο να υπάρχει test suite.

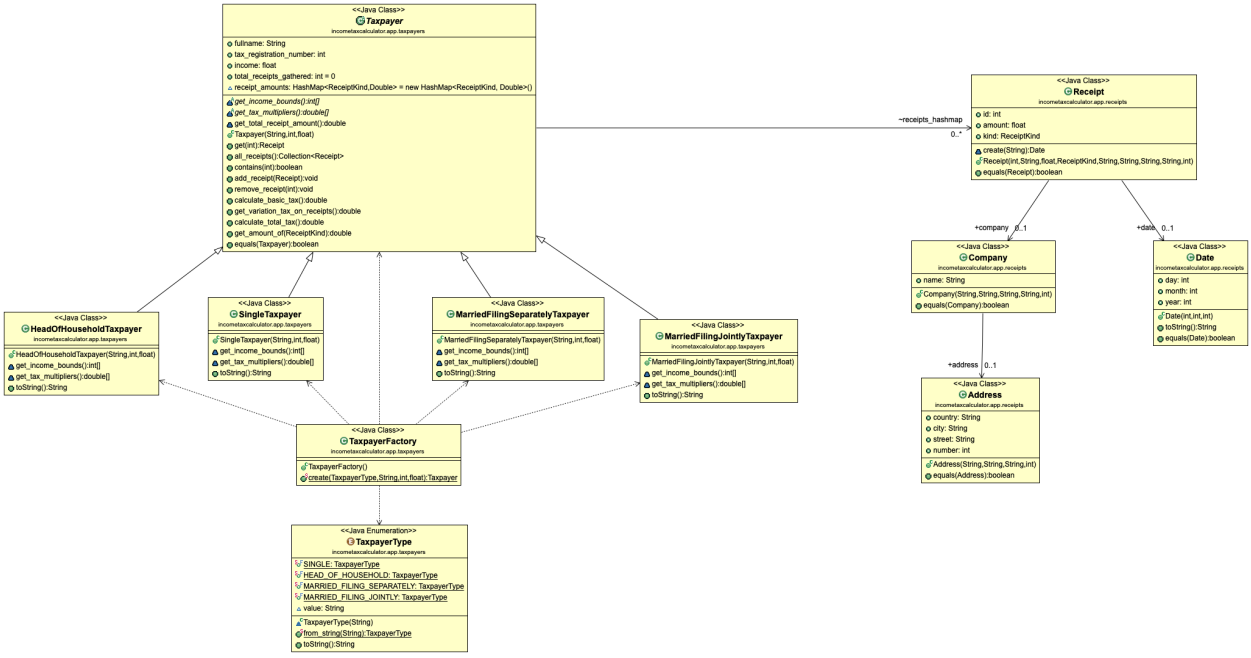
Η τελική μορφή της εφαρμογής παρατίθενται παρακάτω.

ARCHITECTURE

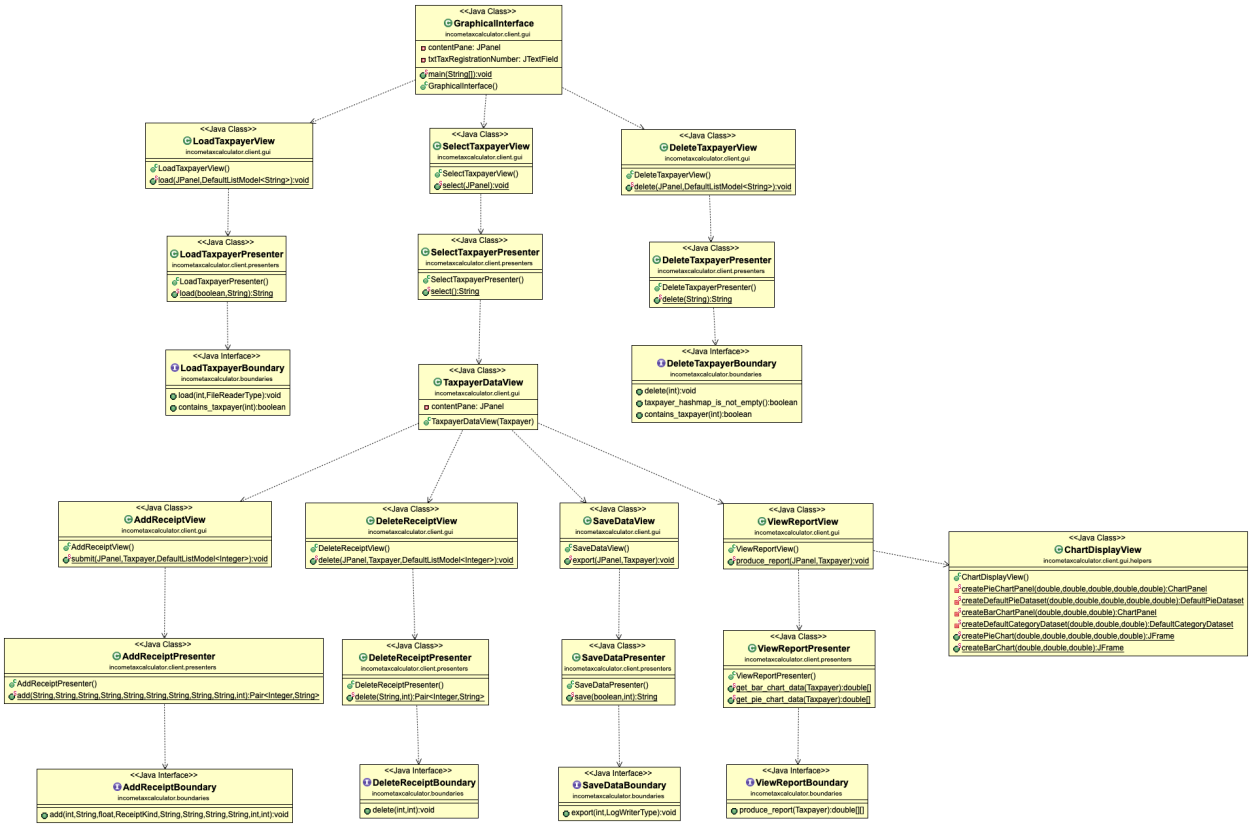
- **Overall diagram**



- **Data relations**



- **Client side**



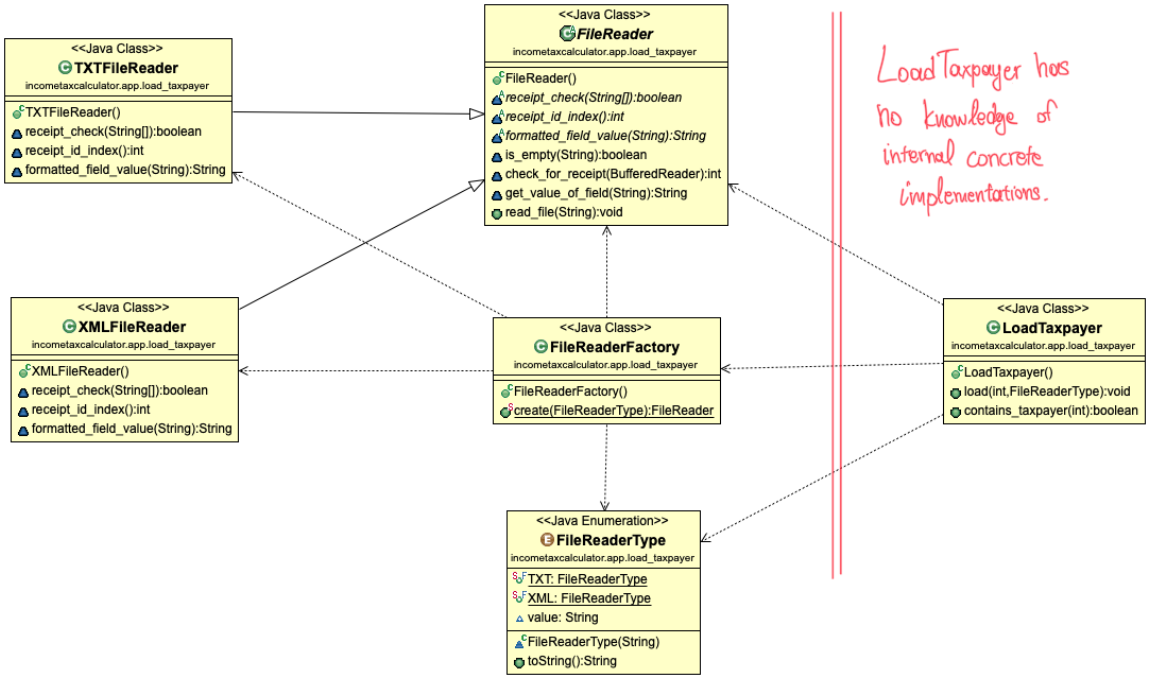
Στο γενικό διάγραμμα είναι φανερές οι σχέσεις μεταξύ Boundaries και Control Objects. Κάθε use case υλοποιείται αναλυτικά. Το **UC5 - Update Information in Files**, λειτουργεί σαν βοηθητικό των **UC3** και **UC4**.

Στο δεύτερο διάγραμμα έχουμε το υποσύστημα δεδομένων. Φαίνεται η κληρονομικότητα και το Factory Pattern για τη δημιουργία Taxpayer. Στο Taxpayer κρατάμε *receipts_hashmap* πεδίο που αποθηκεύει Receipt αντικείμενα.

Στο διάγραμμα του client έχουμε ιεραρχική δομή graphical user interface με κέντρο το αντικείμενο GraphicalInterface. Κάθε view επικοινωνεί με αντίστοιχο presenter. Αυτό επιτρέπει τα view αντικείμενα να έχουν απόλυτο διαχωρισμό από την υπόλοιπη εφαρμογή και τα boundaries με την βοήθεια των buffer αντικειμένων.

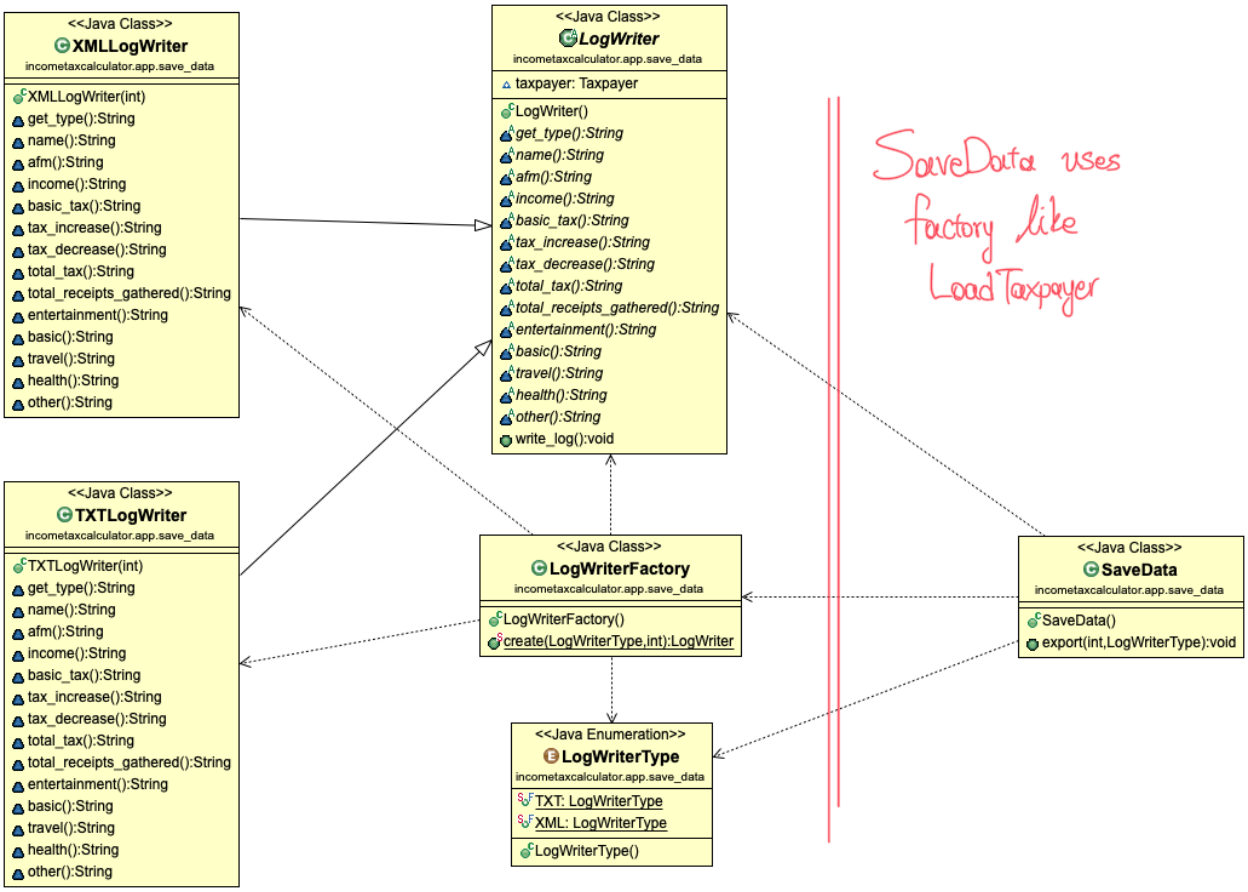
DETAILED DESIGN

- **UC1 - Load Taxpayer**



- **UC5 - Update Information in Files**

- **UC7 - Save Data**



Στο πλαίσιο της εκφώνησης υλοποιήσαμε τις περισσότερες αλλαγές:

(1) **Αφαιρέσαμε accessors** από Entities καθώς τα πεδία χρησιμοποιούνται μόνο από presenters.

(2) Απλοποιήσαμε την Taxpayer χρησιμοποιώντας **κληρονομικότητα**. Κάθε εξειδίκευση της Taxpayer περιέχει λεπτομέρειες και διαφοροποιήσεις όπως **toString** για το όρισμα της οικογενειακής κατάστασης. Η ανάγκη για conditionals αφαιρείται πίσω από την εφαρμογή **αντικειμενοστρέφειας**.

(3) Για αφαίρεση του duplication στα subclasses της Taxpayer κάθε εξειδίκευση περιέχει τα **όρια εισοδήματος** και **πολλαπλασιαστές φόρου** σύμφωνα με τον αλγόριθμο υπολογισμού φόρου.

(4) **Αφαιρέσαμε** τελείως την TaxpayerManager και αναθέσαμε τις λειτουργίες της στα ειδικά πακέτα ακολουθώντας το use case model.

(4.a) Οι λειτουργίες της createTaxpayer() μεταφέρθηκαν στο **UC1** και συγκεκριμένα στην **FileReader**. Για την αφαίρεση των conditionals χρησιμοποιούμε **Factory Pattern** για την δημιουργία ειδικών **Taxpayer**.

(4.b) Οι λειτουργίες της updateFiles() μεταφέρθηκαν στο **UC5**. Οι απαιτήσεις του συστήματος προϋποθέτουν ότι με κάθε ανανέωση αλλάζουν όλοι οι τύποι αρχείων με το ίδιο ΑΦΜ επομένως **δεν** χρησιμοποιήσαμε Factory Pattern.

(4.c) Οι λειτουργίες της saveFiles() μεταφέρθηκαν στο **UC7**. Με **Factory Pattern** κρύψαμε τα conditionals για τη δημιουργία ειδικών **LogWriter**.

(4.d) Οι λειτουργίες της loadTaxpayer() μεταφέρθηκαν στο **UC1**. Χρησιμοποιώντας **Factory Pattern** ομαδοποιήσαμε και κρύψαμε τα conditionals για την δημιουργία ειδικών **FileReader**.

(5) Αφαιρέσαμε διπλότυπα στις TXTFileReader και XMLFileReader **εξειδικεύοντας τις διαφορές** σε μεθόδους και **οργανώνοντας τις ομοιότητες** στην abstract FileReader.

(6) Έχοντας αφαιρέσει την TaxpayerManager η FileWriter χωρίζεται σε **InfoWriter** και **LogWriter** σύμφωνα με τα use cases. Το Refuse Bequest το αποφύγαμε αφαιρώντας όλους τους accessors και κάνοντας public τα πεδία. Η σωστή λύση θα ήταν να χρησιμοποιήσουμε **toString περιγραφές** για τα αντικείμενα αντί να παίρνουμε πρόσβαση στα πεδία ξεχωριστά όταν γράφουμε τα αρχεία, όμως καθώς τα πεδία χρησιμοποιούνται μόνο σε αντικείμενα που συμπεριφέρονται σαν devices (είτε στο I/O είτε στο GUI) υπάρχει η ίδια παραβίαση ενθυλάκωσης που θα είχαν και οι getters πεδίων. Στην περίπτωση της Java μάλιστα, λόγω της στατικής φύσης της γλώσσας και του γεγονότος ότι τα primitives δεν είναι αντικείμενα, η διατήρηση της ενθυλάκωσης είναι πρακτικά αδύνατη. Επομένως καταλήξαμε σε public πεδία.

(7) Τέλος αφαιρέσαμε τα διπλότυπα των **TXTInfoWriter**, **XMLInfoWriter** και **TXTLogWriter** και **XMLLogWriter** χρησιμοποιώντας **Template Pattern** και για τα δύο αντίστοιχα.

CLASSES RESPONSIBILITIES AND COLLABORATIONS (CRC CARDS)

Σύμφωνα με το analysis model, και την BCE αρχιτεκτονική, παρατηρούμε ότι κάθε use case έχει προκαθορισμένο σύνολο κλάσσεων το οποίο εφαρμόζει ανάλογα στην σχεδίαση. Συγκεκριμένα, έχουμε (**Control Object**, **Boundary**, **Presenter**, και **View** για κάθε use case). Η επικοινωνία αναλύθηκε παραπάνω. Πρόσθετα με αυτές τις κλάσσεις έχουμε τα data objects ως **Entities** που είδαμε στο UML διαγραμμα, και βοηθητικές κλάσσεις Pair, και ChartDisplay στο client. Τέλος ώς μορφή αποθήκευσης έχουμε το **TaxpayerHashmap**. Με βάση τα παραπάνω έχουμε τις βασικές CRC περιγραφές:

Class Name: *AddReceipt*	
Responsibilities	**Collaborations**
▪ This class is responsible for adding a new receipt on a Taxpayer's list of receipts.	▪ AddReceiptBoundary ▪ TaxpayerHashmap

Class Name: *DeleteReceipt*	
Responsibilities	**Collaborations**
▪ This class is responsible for deleting a receipt from a Taxpayer's list of receipts.	▪ DeleteReceiptBoundary ▪ TaxpayerHashmap

Class Name: *DeleteTaxpayer*	
Responsibilities	**Collaborations**
▪ This class is responsible for deleting a Taxpayer from TaxpayerHashmap.	▪ DeleteTaxpayerBoundary ▪ TaxpayerHashmap

Class Name: *LoadTaxpayer*	
Responsibilities	**Collaborations**
▪ This class is responsible for loading a new Taxpayer from a file.	▪ LoadTaxpayerBoundary ▪ TaxpayerHashmap ▪ FileReader ▪ FileReaderFactory

Class Name: *SaveData*	
Responsibilities	**Collaborations**
▪ This class is responsible for saving a log about newly added or deleted receipts.	▪ SaveDataBoundary ▪ LogWriter ▪ LogWriterFactory

Class Name: *SelectTaxpayer*	
Responsibilities	**Collaborations**
▪ This class is responsible for selecting a taxpayer from TaxpayerHashmap for display.	▪ SelectTaxpayerBoundary ▪ TaxpayerHashmap

Class Name: *UpdateTaxpayerInformation*	
Responsibilities	**Collaborations**
▪ This class is responsible for selecting a taxpayer from TaxpayerHashmap for display.	▪ UpdateTaxpayerInformationBoundary ▪ InfoWriter ▪ TXTInfoWriter ▪ XMLInfoWriter

Class Name: *ViewReport*	
Responsibilities	**Collaborations**
▪ This class is responsible for gathering taxpayer and receipt information for display.	▪ ViewReportBoundary

TESTING MODEL

Με την ολοκληρωμένη υλοποίηση του Implementation Model, έχουμε ικανοποιήσει τις απαιτήσεις της εφαρμογής. Σε περίπτωση TDD ή BDD έχουμε ήδη γράψει τα unit tests. Στη δική μας περίπτωση στο πλαίσιο της άσκησης ο κώδικας ήταν γραμμένος επομένως το refactoring έγινε χωρίς test. Ιδανικά θα έπρεπε να υπάρχουν όμως για την επίλυση επιλέξαμε να χρησιμοποιήσουμε τα αρχικά acceptance tests.

Εφόσον έχουμε ελέγξει κάθε μέθοδο της εφαρμογής (προσεγγίζοντας 100% coverage), ξεκινάμε την εγγραφή των integration tests. Αυτά θα επιβεβαιώσουν συνδυαστικά την επικοινωνία πολλών υποσυστημάτων του κώδικα. Μερικά από τα unit tests μπορούν να εξυπηρετήσουν τη μορφή integration tests.

Στο πακέτο `test` έχουμε το σύνολο ελέγχων. Χρησιμοποιήσαμε junit4.

TODO

(1) Αφαίρεση όλων των conditionals σε διάφορες κλάσσεις της εφαρμογής (client μπορεί να εχεί conditionals χωρίς ιδιαίτερη επιρροή στην αρχιτεκτονική).

(2) Απλοποίηση της κληρονομικότητας στους FileReader. Έχουμε ειδικές μεθόδους χωρίς καλή περιγραφή της εξειδίκευσης τους στα ονόματα.

(3) Διόρθωση του SelectTaxpayerPresenter. Έχουμε dependencies μεταξύ Presenter και Swing.

(4) Υλοποίηση acceptance test για use cases με I/O.

(5) Λείπει μεγάλο κομμάτι των unit test για ικανοποιητικό coverage.

(6) Υλοποίηση integration tests.

(7) Αφαίρεση του FileWriter ως interface από InfoWriter και LogWriter υλοποιώντας μία write μέθοδο.

(8) Μεταφορά των conditionals του Taxpayer όσο πιο κοντά στα boundaries γίνεται.

(9) Χρήση Gateway Pattern για το persistence του TaxpayerHashmap.