Πανεπιστήμιο Κρήτης –Τμήμα Επιστήμης Υπολογιστών

ΗΥ252– Αντικειμενοστρεφής Προγραμματισμός

Διδάσκων: Ι. Τζίτζικας

Χειμερινό Εξάμηνο 2020-2021

[ΗΜΕΡΑ ΠΛΗΡΩΜΗΣ]

ΓΙΟΒΑΝΟΠΟΥΛΟΣ ΝΙΚΟΛΑΟΣ

CSD4613

9/01/2022

Περιεχόμενα

1. Εισαγωγή

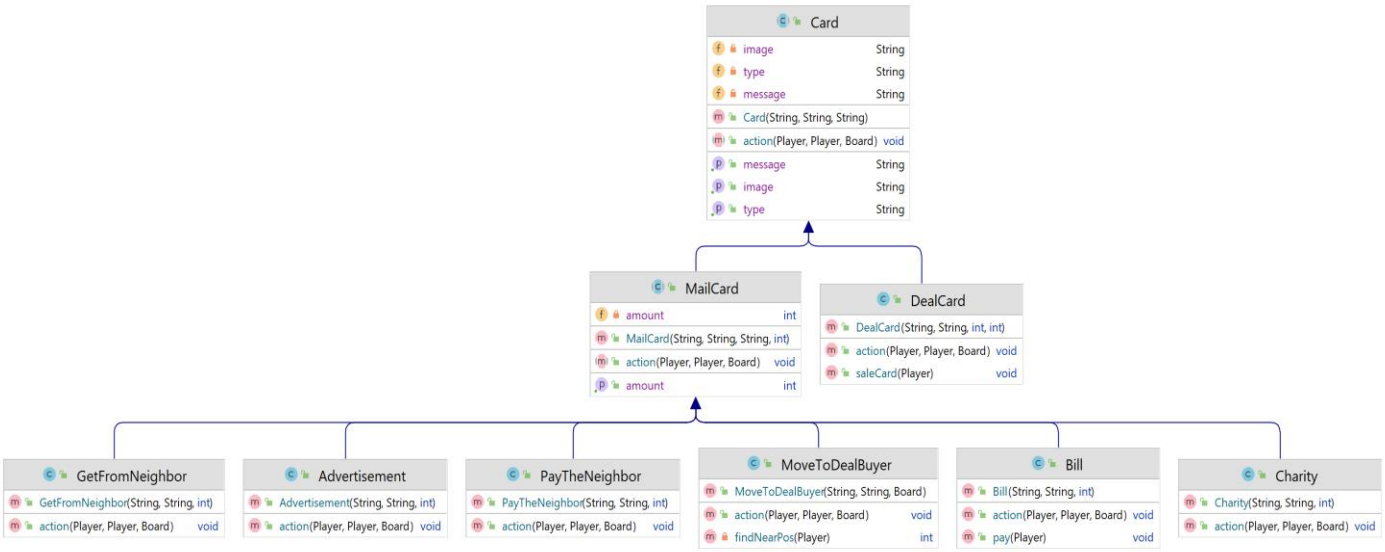
Για την εργασία μου χρησιμοποίησα το μοντέλο MVC (Model View Controller). Πρόκειται για μια αρχιτεκτονική λογισμικού κατά την οποία το πρόγραμμα χωρίζεται σε 3 μέρη, Το Model που είναι ο πυρήνας του προγράμματος, το View που αφορά την γραφική απεικόνιση του προγράμματος και τέλος το Controller, το οποίο αποτελεί την γέφυρα μεταξύ των 2 και ρόλος του είναι να τα διευθύνει, καθώς επίσης και όλη την ροή του προγράμματος. Στις υπόλοιπες ενότητες θα δούμε αναλυτικά αυτά τα 3 πακέτα και ό,τι αυτά περιλαμβάνουν καθώς και κάποια επεξηγηματικά διαγράμματα UML για βαθύτερη κατανόηση.

2. Η Σχεδίαση και οι Κλάσεις του Πακέτου Model

Στο εν λόγω πακέτο περιέχονται 6 διαφορετικά πακέτα τα οποία θα αναλυθούν περεταίρω στην συνέχεια. Συγκεκριμένα περιέχονται τα πακέτα : Board, Card, Dice, Player, Position και Tile.

Πακέτο Card

Σε αυτό το πακέτο περιέχεται η abstract κλάση “Card” που υλοποιεί την έννοια της κάρτας. Στην συνέχεια μια κάρτα εξειδικεύεται είτε σε κάρτα μηνύματος, είτε σε κάρτα συμφωνίας. Για αυτό στην συνέχεια ακολουθούν οι κλάσεις “MailCard” (abstract) και “DealCard” οι οποίες κληρονομούν την κλάση “Card”. Από την εκφώνηση τώρα, γνωρίζουμε ότι υπάρχουν 6 είδη καρτών μηνύματος. Έτσι, λοιπόν, ακολουθούν οι κλάσεις “Advertisement”, “Bill”, “Charity”, “GetFromNeighbor”, “MoveToDealBuyer”,”PayTheNeighbor”, οι οποίες προφανώς κληρονομούν την κλάση “Mailcard”. Μπορούμε τις σχέσεις αυτές να τις δούμε και διαγραμματικά με την παρακάτω διάγραμμα UML.



Θα δούμε τώρα αναλυτικά τα attributes και τις μεθόδους κάθε κλάσης.

Class Card:

Κάνει implement την Serializable.

Attributes:

```
private String type;        // Ο τύπος μίας κάρτας π.χ. Deal, Advertisement, κλπ.
private String image;       // Το όνομα της εικόνας που έχουμε αποθηκεύσει στον
                            // φάκελο resources π.χ petreleo.png
private String message;     // Το μήνυμα που θα εμφανιστεί στην οθόνη όταν ο χρήστης
                            // τραβήξει την κάρτα π.χ. «Κέρδισες 50 Ευρώ».
```

Methods:

```
public String getType() {return type;}
```

· Accessor συγκεκριμένα getter.

```
public String getImage() {return image;}
```

· Accessor συγκεκριμένα getter.

```
public String getMessage() {return message;}
```

· Ομοίως με τα παραπάνω.

public Card(String type, String image, String message)

· Constructor της κλάσης, ο οποίος αναθέτει τις τιμές των attributes της κλάσης.

public abstract void action(Player p1, Player p2, Board b);

· Transformer. Πρόκειται για την μέθοδο με την οποία κάθε κάρτα θα εκτελεί τις ενέργειες της π.χ. να μεταφέρει λεφτά σε κάποιον παίχτη. Οι παράμετροι τύπου Player και Board θα αναλυθούν παρακάτω. Προς το παρόν αρκεί να γνωρίζουμε πως οι κάρτες πρέπει να επεξεργάζονται τα δεδομένα τόσο των παιχτών του παιχνιδιού, όσο και του ταμπλού, παραδείγματος χάρη το Jackpot ανήκει στο ταμπλό και ορισμένες φορές ένας παίχτης αναγκάζεται να καταθέσει λεφτά σε αυτό λόγω ορισμένης κάρτας. Ο λόγος που ορίζουμε abstract την συγκεκριμένη μέθοδο είναι επειδή κάθε παιδί/εγγόνι της κλάσης Card χρειάζεται να εκτελέσει μία ενέργεια, επομένως κάθε τύπος κάρτα θα χρειαστεί την δική του μέθοδο action.

✯ Σημείωση ✯ : Αξίζει να σημειωθεί εδώ, πως δεν χρειάζονται όλες οι κάρτες όταν εκτελούν την ενέργεια τους και τους 2 παίχτες (player) του παιχνιδιού ή και το ταμπλό (board). Ωστόσο, επιλέγουμε να συμπεριλάβουμε και τα ως παραμέτρους της μεθόδου action , ακόμη και αν κάποια δεν χρειαστεί να τα χρησιμοποιήσουμε, προκείμενου να κρατήσουμε μια ομοιομορφία στην μέθοδο action και στην συνέχεια, όταν θα χρειαστεί να καλέσουμε την μέθοδο action για μια κάρτα η οποία δημιουργήθηκε με Dynamic Binding (π.χ. Card a = new DealCard();) να μην χρειάζεται να παίρνουμε περιπτώσεις για το τι τύπος κάρτας είναι ώστε να περάσουμε τα αντίστοιχα ορίσματα. Αυτό θα γίνει σαφέστερο στην συνέχεια.

Class MailCard:

Κάνει extend την κλάση Card.

Attributes:

private int amount; //Το ποσό το οποίο ο παίχτης πρέπει να λάβει/πληρώσει

Αξίζει να σημειωθεί πως η μόνη κάρτα που δεν χρειάζεται το πεδίο amount είναι μία εξειδίκευση της MailCard, η MoveToDealBuyer. Ωστόσο όλες οι υπόλοιπες κάρτες το χρειάζονται, οπότε το λογικό είναι απλώς να μην αξιοποιηθεί από την MoveToDealBuyer

Methods:

public int getAmount() {return amount;}

· Accessor, συγκεκριμένα getter.

public MailCard(String type, String image, String message,int amount)

· Constructor της κλάσης, αναθέτει τις τιμές των attributes (καλώντας και τον constructor της μαμάς κλάσης).

public void action(Player p1, Player p2, Board b)

· Ισχύουν τα ίδια με την κλάση Card.

✯ Σημείωση ✯ : Αν και προφανές, ο λόγος που ορίζουμε abstract την κλάση MailCard είναι επειδή μία MailCard δεν μπορεί να υπάρξει από μόνη της αφού πρέπει να καθοριστεί τι τύπος MailCard είναι πχ. Advertisement, Charity κλπ. σε αντίθεση με τις κάρτες συμφωνίας – DealCards, οι οποίες δεν εξειδικεύονται περεταίρω.

Class DealCard:

Κάνει επίσης extend την κλάση Card.

Attributes:

private int costPrice; //αφορά την τιμή αγοράς της κάρτας.

private int salePrice; //αφορά την τιμή πώλησης της κάρτας αφού αγοραστεί.

Methods:

public DealCard(String image, String message, int costPrice, int salePrice)

· Constructor της κλάσης.

public void saleCard(Player p)

· Transformer. Είναι η μέθοδος που πουλάει την κάρτα. Θα την χρειαστεί η κλάση BuyerPosition που θα δούμε σε λίγο. Στο block της χρησιμοποιεί την μέθοδο setWallet του αντικειμένου p η οποία ρυθμίζει τα λεφτά που θα πάρει ο παίχτης απτή πώληση της εν λόγω κάρτας.(Θα την δούμε στην κλάση Player)

@Override

public void action(Player p1, Player p2, Board b)

· Εδώ βλέπουμε την πρώτη υλοποίηση της action! Εδώ πέρα οι Player p2, Board b είναι περιττοί.(βλέπε σελ. 3 - ✯ Σημείωση ✯). Εδώ θα δίνεται η επιλογή στον παίχτη (αντικείμενο p1) να αγοράσει η να αγνοήσει την κάρτα συμφωνίας. Αν επιλέξει να την αγοράσει θα πρέπει να γίνεται έλεγχος για το αν τα χρήματα του (wallet) επαρκούν και αν όχι να παίρνει δάνειο ώστε να επαρκούν. Αυτό θα γίνεται με την μέθοδο της κλασης Player : walletcheck(int amount). Έπειτα να ενημερώνουμε το πορτοφόλι του αφού για την αγορά του «πήραμε κάποια χρήματα» και τέλος να προσθέτουμε την κάρτα στην συλλογή του.Αυτο θα γίνεται με την μέθοδο addDealCard() της κλάσης Player. Όλες αυτές τις μεθόδους θα τις δούμε στην συνέχεια στην κλάση Player.

Class Advertisement:

Κάνει extend την κλάση MailCard. Δεν προστίθεται κάποιο καινούριο attribute. Οι μέθοδοί της είναι ο constructor της (ίδια ορίσματα με base class της) και η μέθοδος action.

public Advertisement(String image, String message,int amount)

@Override

public void action(Player p1, Player p2, Board b)

· Εδώ απλώς χρειάζεται να δώσουμε το amount (το οποίο θα είναι 20) στον Player p1. Θα το πετύχουμε όπως είδαμε και παραπάνω με την μέθοδο setWallet. Τα Player p2, Board b εδώ είναι περιττά.

Class Bill:

Κάνει extend την κλάση MailCard. Δεν προστίθεται κάποιο καινούριο attribute.Ακολουθούν οι μέθοδοι της :

public Bill(String image, String message,int amount)

· Constructor της κλάσης.

@Override

public void action(Player p1, Player p2, Board b)

· Εδώ θα χρειαστεί να χρησιμοποιήσουμε τις μεθόδους του αντικειμένου p1 : addPayCard για να θυμόμαστε ποια κάρτα πρέπει να πληρώσει στο τέλος ο παίχτης και την setBills η οποία προσθέτει το πόσο που θα χρειαστεί να πληρώσει ο παίχτης στο τέλος του μήνα. Όπως και πριν τα Player p2, Board b είναι περιττά και δεν χρησιμοποιούνται.

public void pay(Player p)

· Πρόκειται για την μέθοδο που θα χρησιμοποιήσει αργότερα η κλάση PayDayPosition προκειμένου να πάρει τα λεφτά που χρωστάει ο παίκτης σε λογαριασμούς. Για την επίτευξη αυτού χρησιμοποιούνται μεθόδοι της κλάσης Player όπως είδαμε και προηγουμένως.

Class Charity:

Κάνει extend την κλάση MailCard. Δεν προστίθεται κάποιο καινούριο attribute.Ακολουθούν οι μέθοδοι της :

public Charity(String image, String message,int amount)

· Constructor της κλάσης.

@Override

public void action(Player p1, Player p2, Board b)

· Εδώ ο Player p2 είναι περιττός. Εδώ χρησιμοποιούμε μεθόδους για απόσπαση χρημάτων από από τον player όπως είδαμε και προηγουμένως ΚΑΙ χρησιμοποιούμε επίσης την μέθοδο setJackpot και getJackpot της κλάσης Board προκειμένου να μεταφέρουμε τα λεφτά του παίχτη (amount) στο jackpot του επιτραπέζιου. Θα γίνει πιο ξεκάθαρη στην ανάλυση της κλάσης Board.

Class GetFromNeighbor:

Κάνει extend την κλάση MailCard. Δεν προστίθεται κάποιο καινούριο attribute.Ακολουθούν οι μέθοδοι της :

public GetFromNeighbor(String image, String message,int amount)

· Constructor της κλάσης.

@Override

public void action(Player p1, Player p2, Board b)

· Εδώ χρησιμοποιούμε το p1 και p2, ενώ το b είναι περιττό. Μέσω μεθόδων της κλάσης Player που χρησιμοποιήσαμε και προηγουμένως, 1) Ελέγχουμε αν το ποσό που πρέπει να δώσει ο p2 στον p1 είναι διαθέσιμο, αλλιώς δίνουμε δάνειο στον p2, 2) αυξάνουμε το πορτοφόλι του P1 με το ποσό (getAmount()) , 3) μειώνουμε το πορτοφόλι του p2 με το ποσό.

Class PayTheNeighbor:

Κάνει extend την κλάση MailCard. Δεν προστίθεται κάποιο καινούριο attribute.Ακολουθούν οι μέθοδοι της :

public PayTheNeighborString image, String message,int amount)

· Constructor της κλάσης.

@Override

public void action(Player p1, Player p2, Board b)

· Ακριβώς ίδια υλοποίηση με αυτήν της GetFromNeighbor με την διαφορά ότι τα arguments p1 p2 χρησιμοποιούνται ανάποδα(προφανές).

Class MoveToDealBuyer:

Κάνει extend την κλάση MailCard. Προστίθεται ένα καινούριο attribute μαζί με τα παλιά :

private Board board; // περιέχει όλο το επιτραπέζιο, Ο λόγος είναι για να έχουμε πρόσβαση στην τοποθεσία των θέσεων τύπου Buyer προκειμένου να μετακινήσουμε τον παίχτη που τράβηξε την κάρτα στην κοντινότερη.

Μέθοδοι:

public MoveToDealBuyer(String image, String message, Board board)

· Constructor της κλάσης.

private int findNearlPos(Player p)

· Μέθοδος λαμβάνει τον player,παίρνει την τοποθεσία του με την getPosition() (θα την δούμε αργότερα) και με βάση αυτή βρίσκει το πλακίδιο που είναι τύπου Buyer/Deal (εξού και το attribute Board board) από το ταμπλό και επιστρέφει το Index του. π.χ 12 για 12ο πλακίδιο κλπ. Αν δεν υπάρχει προς τα μπροστά κανένα τέτοιο πλακίδιο επιστρέφει 0.

@Override

public void action(Player p1, Player p2, Board b)

· Εδώ καλούμε την findNearDeal(p1) προκειμένου να πάρουμε το index Που θα χρειαστεί να μετακινήσουμε τον παίχτη. Αν, λοιπόν, το index δεν είναι μηδέν τότε με την μέθοδο setPosition μετακινούμε τον p1 (Θα την δούμε στην κλάση Player).

Πακέτο Player

Ήρθε, λοιπόν, η ώρα να αναλύσουμε το πακέτο Player, στο οποίο περιέχεται η κλάση Player που αναφέραμε τόσες πολλές φορές.

Class Player:

Κάνει implement την Serializable.

Attributes:

```
private int wallet;          //Τα λεφτά που κατέχει ο κάθε παίχτης
private int numOfPaydays;    //Πόσες φορές έχει φτάσει στην ημέρα Payday(31η ημέρα)
private int bills;           //Το συνολικό ποσό που ο παίχτης χρωστάει από κάρτες
                             //τύπου Bill.
private int debts;           //Το συνολικό ποσό δανείων του παίχτη
private int playerNumber;    //Αριθμός παίχτη π.χ. 1ος, 2ος
private ArrayList<DealCard> cardsOwned = new ArrayList<DealCard>();
```

//Arraylist που περιέχει όλες τις κάρτες τύπου Deal που ο παίχτης αγόρασε και δεν πούλησε ακόμα

```
private ArrayList<MailCard> cardsToPay = new ArrayList<MailCard>();
```

//Arraylist Που περιέχει όλες τις κάρτες που πρέπει να πληρώσει στο τέλος του μήνα όπως οι κάρτες τύπου Bill που είδαμε προηγουμένως.

```
public Dice dice;
```

//Αντικείμενο τύπου Dice, κλάση που θα μελετήσουμε αργότερα η οποία υλοποιεί το ζάρι που θα χρησιμοποιεί κάθε παίχτης.

```
private PlayerPosition position;
```

// Αντικείμενο τύπου PlayerPosition που κληρονομεί την κλάση Position.Θα αφιερώσουμε ολόκληρο πακέτο στο Position, υπομονή 😊. Σημασία έχει πως το εν λόγω attribute κρατά την τοποθεσία του παίχτη επάνω στο ταμπλό π.χ. βρίσκεται στην μέρα 5η, 14η κλπ.

Μέθοδοι:

```
public Player(int playerNumber)
```

· Constructor της κλάσης.

```
public ArrayList<MailCard> getCardsToPay()
```

· Getter, επιστρέφει το ArrayList CardsToPay.

```
public int getNumOfPaydays()
```

· Getter, επιστρέφει το πεδίο NumOfPaydays.

```
public void updateNumofPaydays()
```

· Transformer, κάνει increment το πεδίο NumofPaydays.Θα την χρησιμοποιήσουμε στην κλάση PayDayPosition αργότερα.

```
public int getPlayerNumber()
```

· Getter, επιστρέφει το πεδίο PlayerNumber.

```
public void setPosition(int boardNumber, String day)
```

· Setter, αναθέτει τιμή στο πεδίο position.

```
public int getPosition()
```

· Getter, επιστρέφει το πεδίο boardNumber του πεδίου position της κλάσης μας (Θα γίνει πιο κατανοητό στο πακέτο Position).

public String getposition()

· Getter, επιστρέφει το πεδίο day του πεδίου position της κλάσης μας (Θα γίνει πιο κατανοητό στο πακέτο Position).

public int getWallet()

· Getter, επιστρέφει το πεδίο wallet.

public void setWallet(int wallet)

· Setter, αναθέτει τιμή στο πεδίο wallet.

public int getDebts()

· Getter, επιστρέφει το πεδίο debts.

public void setDebts

· Setter, αναθέτει τιμή στο πεδίο debts.

public ArrayList<DealCard> getCardsOwned()

· Getter, επιστρέφει το πεδίο cardsOwned.

public void walletCheck(int amount)

· Transformer/Observer, ελέγχει αν το amount είναι μικρότερο από το wallet. Αν δεν είναι παίρνει δάνειο πολλαπλάσιο του 1000. Διαφορετικά δεν κάνει τίποτα. Αυτό επιτυγχάνεται με τις μεθόδους getWallet, setWallet, setDebts, getDebts.

public void addPayCard(MailCard mc)

· Transformer, προσθέτει την κάρτα mc στο Araaylist cardsToPay

public void setBills(int bills)

· Setter, αναθέτει τιμή στο πεδίο bills.

public int getBills()

· Getter, επιστρέφει το πεδίο bills.

public Card popPayCard()

· Transformer, κάνει pop μία κάρτα από το Araaylist cardsToPay.

public void addDealCard(DealCard dc)

· Transformer, προσθέτει την κάρτα dc στο Araaylist cardsOwned.

public Card popDealCard(int index)

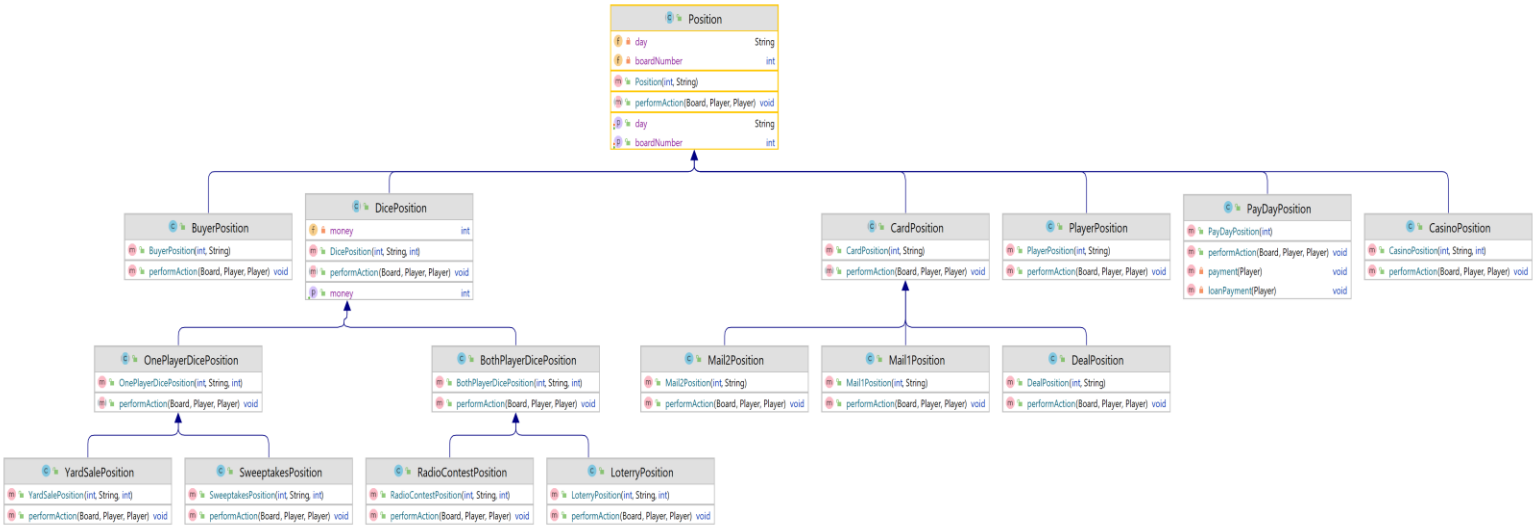
· Transformer, κάνει pop την κάρτα με index=index από το Araaylist cardsOwned.

Όλα αυτά τα χαρακτηριστικά που περιγράψαμε παραπάνω για την κλάση Board, μπορούμε να τα απεικονίσουμε και με ένα διάγραμμα UML :

Πακέτο Position

Τώρα σειρά έχει η ανάλυση του πακέτου Position. Σε αυτό περιέχεται η abstract κλάση Position, η οποία έχει σκοπό να υλοποίηση την έννοια της τοποθεσίας, μιλώντας πάντα για ένα επιτραπέζιο παιχνίδι. Τώρα, μια τοποθεσία μπορεί να αφορά είτε έναν παίχτη, είτε ένα σημείο του ταμπλού. Στην συνέχεια ένα σημείο του ταμπλού μπορεί να ανήκει σε μια από τις παραπάνω 9 κατηγορίες : Θέση μηνύματος, Θέση Συμφωνίας, Θέση Λαχείο, Θέση Λοταρίας, Θέση Διαγωνισμός στο Ραδιόφωνο, Θέση Αγοραστής, Θέση Βραδιά Οικογενειακού Καζίνο, Θέση Αγορά με Έκπτωση και Θέση Ημέρα Πληρωμής. Για να μπορέσουμε, λοιπόν να δομήσουμε καθαρά και κατανοητά όλες αυτές τις έννοιες και να μπορέσουμε να επαναχρησιμοποιήσουμε όσο το δυνατόν περισσότερο κώδικα, ακολουθούμε την εξής ιεραρχία :

Abstract κλάση Position. Στην συνέχεια ακολουθεί η κλάση PlayerPosition, η κλάση DicePosition, η κλάση CardPosition, CasinoPosition και το BuyerPosition, οι οποίες κληρονομούν την κλάση Position. Τώρα η CardPosition έχει τις κλάσεις : DealPosition, Mail1Position και Mail2Position οι οποίες την κληρονομούν. Μετά για την DicePosition έχουμε την OnePlayerDicePosition η οποία την κληρονομεί και αυτή με την σειρά της κληρονομείται από την BothPlayerDicePosition. Τώρα με την σειρά της η OnePlayerDicePosition κληρονομείται από τις κλάσεις : YardSalePosition και SweeptakesPosition, Τέλος, η BothPlayerDicePosition κληρονομείται από τις κλάσεις : RadioContestPosition και LoterryPosition. Μπορούμε τις σχέσεις αυτές να τις δούμε και διαγραμματικά με την παρακάτω διάγραμμα UML.



Πάμε τώρα να δούμε μία-μία αναλυτικά.

Class Position:

Κάνει implement την Serializable.

Abstract κλάση με πεδία :

```
private int boardNumber;     //αριθμός ταμπλό/μέρα του μήνα
private String day;          //Μέρα που αντιστοιχεί στο ταμπλό
```

Μέθοδοι:

```
public Position(int boardNumber, String day)
```

· Constructor της κλάσης.

```
public int getBoardNumber()
```

· Accessor, συγκεκριμένα Getter, επιστρέφει το πεδίο boardNumber.

```
public String getDay()
```

· Accessor, συγκεκριμένα Getter, επιστρέφει το πεδίο Day της κλάσης μας.

```
public void setBoardNumber(int boardNumber)
```

· Setter, αναθέτει τιμή στο πεδίο boardNumber της κλάσης μας.

```
public void setDay(String day)
```

· Setter, αναθέτει τιμή στο πεδίο Day.

```
public abstract void  performAction(Board b, Player p1, Player p2)
```

· Πρόκειται για την μέθοδο που θα καλείται μόλις ένας παίχτης πέσει πάνω σε οποιοδήποτε position. Κάθε derived κλάση της Position, λοιπόν, θα χρειαστεί αυτή την μέθοδο γι'αυτό την δηλώνουμε abstract.

✯ Σημείωση ✯ : Όπως είχαμε δει και στην κλάση Card και νομίζω θα γίνει απόλυτα κατανοητό στην υλοποίηση της κλάσης Controller, είναι πολύ βολικό να μπορούμε να καλούμε την μέθοδο PerformAction χωρίς να γνωρίζουμε/παίρνουμε περιπτώσεις για το σε ποιά derived κλάση ανήκει το στιγμιότυπο από το οποίο την καλούμε. Έτσι κρατούμε σε όλες τις κλάσεις τις ίδιες παραμέτρους στην performAction, αγνοώντας τα περιττά arguments και δουλεύοντας σε κάθε περίπτωση με αυτά που χρειαζόμαστε.

Class PlayerPosition:

Κάνει extend την κλάση Position. Δεν προστίθεται κάποιο καινούριο attribute.Ακολουθούν οι μέθοδοι της :

public PlayerPosition

· Constructor της κλάσης.

@Override

public void performAction(Board b, Player p1, Player p2)

· Η μέθοδος performAction αφορά positions του ταμπλού και όχι του Player. Ωστόσο επειδή και ο Player χρειάζεται position και θέλουμε να κάνουμε επαναχρησιμοποίηση κώδικα κάνουμε extend την Position και επομένως αναγκαζόμαστε να κάνουμε και Override την μέθοδο performAction. Αυτό φυσικά δεν αποτελει πρόβλημα καθώς μπορούμε να την αφήσουμε κενή μιας και δεν την χρειαζόμαστε και δεν θα την καλέσουμε ποτέ ούτως ή άλλως.

Class DicePosition:

Abstract κλάση που κάνει extend την κλάση Position. Προστίθεται ένα καινούριο attribute μαζί με τα παλιά :

```
private int money;    //Το χρηματικό έπαθλο του νικητή των ζαριών
```

Ακολουθούν οι μέθοδοι :

public DicePosition(int boardNumber, String day,int money)

· Constructor της κλάσης.

public int getMoney()

· Accessor, συγκεκριμένα Getter, επιστρέφει το πεδίο money της κλάσης.

public abstract void performAction(Board b, Player p1, Player p2)

· Η μέθοδος performAction που είδαμε και παραπάνω. Είναι abstract, κοινώς δεν έχει υλοποίηση.

Class CardPosition:

Abstract κλάση που κάνει extend την κλάση Position. Δεν προστίθεται κάποιο καινούριο attribute.Ακολουθούν οι μέθοδοι της :

public CardPosition(int boardNumber, String day)

· Constructor της κλάσης.

public abstract void performAction(Board b, Player p1, Player p2)

· Η μέθοδος performAction που είδαμε και παραπάνω. Είναι abstract, κοινώς δεν έχει υλοποίηση.

Class CasinoPosition:

Κάνει extend την κλάση Position. Προστίθεται ένα καινούριο attribute μαζί με τα παλιά :

private int amount; //τα χρήματα που ο παίκτης θα κερδίσει/χάσει

Ακολουθούν οι μέθοδοι :

public CasinoPosition(int boardNumber, String day,int amount)

· Constructor της κλάσης.

@Override

public void performAction(Board b, Player p1, Player p2)

· Εδώ βλέπουμε την πρώτη υλοποίηση της performAction! Η μόνη παράμετρος που μας χρειάζεται είναι η Player p1, οι υπόλοιπες είναι περιττές, (για τον λόγο που περιλαμβάνονται βλέπε σελ. 11 - ✯ Σημείωση ✯).Η κλάση dice που θα δούμε σε επόμενο πακέτο περιέχει το attribute last που κρατά την τελευταία ζαριά. Έτσι παίρνουμε με τον αντίστοιχο getter την τελευταία ζαριά και εξετάζουμε μονό η ζυγό αριθμό ο παίχτης που έπεσε πάνω σε αυτήν την θέση. Ανάλογα με το αποτέλεσμα προσθέτουμε/αφαιρούμε το amount από τον p1 μέσω των μεθόδων setWallet/getWallet/walletCheck.

Class BuyerPosition:

Κάνει extend την κλάση Position. Δεν προστίθεται κάποιο καινούριο attribute. Ακολουθούν οι μέθοδοι της :

public BuyerPosition(int boardNumber, String day)

· Constructor της κλάσης.

@Override

public void performAction(Board b, Player p1, Player p2)

· Ομοίως με την CasinoPosition, η μόνη παράμετρος που μας χρειάζεται είναι η Player p1, οι υπόλοιπες είναι περιττές. Εδώ ελέγχουμε αν ο p1 έχει αγορασμένες κάρτες συμφωνίας. Αν έχει θα χρειαστεί να ανοίξουμε ένα dialog box που να του δίνει την επιλογή να διαλέξει ποια κάρτα θέλει να πουλήσει. Αυτό το επιτυγχάνουμε την κλάση SellCardWindow του πακέτου View που θα το δούμε πολύ αργότερα. Μόλις, λοιπόν, λάβουμε την επιλογή κάνουμε pop την κάρτα από το Arraylist του P1 και κάνουμε την ενέργεια salecard (μέθοδοι που είδαμε παραπάνω).

Class DealPosition:

Κάνει extend την κλάση CardPosition. Δεν προστίθεται κάποιο καινούριο attribute. Ακολουθούν οι μέθοδοι της :

public DealPosition(int boardNumber, String day)

· Constructor της κλάσης.

@Override

public void performAction(Board b, Player p1, Player p2)

Εδώ οι παράμετροι που θα χρειαστούμε είναι : Board b και Player p1 (Player p2 περιττή). Προκειμένου να τραβήξει μία κάρτα συμφωνίας ο παίχτης που έπεσε σε αυτή τη θέση, θα χρειαστούμε την μέθοδο drawOneCard της κλάσης Board (θα την δούμε στο πακέτο Board), με την οποία τραβάμε μία τυχαία κάρτα συμφωνίας (βάζοντας argument επιλογής για dealcard) προκειμένου να καλέσουμε την μέθοδο action για να ενεργοποιηθεί η κάρτα.

Class Mail1Position:

Κάνει extend την κλάση CardPosition. Δεν προστίθεται κάποιο καινούριο attribute. Ακολουθούν οι μέθοδοι της :

public Mail1Position(int boardNumber, String day)

· Constructor της κλάσης.

@Override

public void performAction(Board b, Player p1, Player p2)

· Ακριβώς ίδια διαδικασία με αυτή της κλάσης DealPosition με την διαφορά ότι κάνουμε την drawOneCard να τραβήξει κάρτα μηνύματος και όχι συμφωνίας.

Class Mail2Position:

Πρόκειται για μία κλάση η οποία είναι πανομοιότυπη με την Mail1Position. Ο λόγος που την δημιουργούμε είναι καθαρά για δική μας ευκολία στην αρχικοποίηση και στην διαχείριση των καρτών αργότερα (βλέπε κλάση Board/Controller).

Class OnePlayerDicePosition:

Abstract κλάση ηοποία κάνει extend την κλάση DicePosition. Δεν προστίθεται κάποιο καινούριο attribute. Ακολουθούν οι μέθοδοι της :

public OnePlayerDicePosition(int boardNumber, String day,int money)

· Constructor της κλάσης.

public abstract void performAction(Board b, Player p1, Player p2)

· Είναι abstract, κοινώς δεν έχει υλοποίηση.

Class BothPlayerDicePosition:

Κάνει extend την κλάση DicePosition. Δεν προστίθεται κάποιο καινούριο attribute. Ακολουθούν οι μέθοδοι της :

public BothPlayerDicePosition(int boardNumber, String day,int money)

· Constructor της κλάσης.

@Override

public void performAction(Board b, Player p1, Player p2)

· Εδώ χρειαζόμαστε τα p1, p2 (Board b περιττό) καθώς και την κλάση DiceWindow του πακέτου View (θα το δούμε αργότερα) προκειμένου να ανοίξουμε ένα dialog box στους παίχτες και με την σειρά να ρολάρουν το ζάρι τους. Σε περίπτωση ισοπαλίας επαναλαμβάνουμε. Ανακοινώνουμε τον νικητή και του καταθέτουμε το χχρηματικό του έπαθλο (getMoney()).

Class YardSalePosition:

Κάνει extend την κλάση OnePlayerDicePosition. Δεν προστίθεται κάποιο καινούριο attribute. Ακολουθούν οι μέθοδοι της :

public YardSalePosition(int boardNumber, String day,int money)

· Constructor της κλάσης.

@Override

public void performAction(Board b, Player p1, Player p2)

· Χρειαζόμαστε μόνο το p1 και b. Ακολουθούμε την ίδια διαδικασία με την BothPlayerDicePosition, μόνο που γίνεται για έναν παίχτη προφανώς, και στο τέλος τραβάμε και μία κάρτα συμφωνίας με την drawOneCard της κλάσης Board.

Class SweeptakesPosition:

Κάνει επίσης extend την κλάση OnePlayerDicePosition. Δεν προστίθεται κάποιο καινούριο attribute. Ακολουθούν οι μέθοδοι της :

public SweeptakesPosition(int boardNumber, String day,int money)

· Constructor της κλάσης.

@Override

public void performAction(Board b, Player p1, Player p2)

· Ακολουθούμε παρόμοια διαδικασία με τις προηγουμενες 2 κλάσεις στο θέμα υλοποήσης. Η διαφορά είναι στο πως υπολογίζεται το ποσό που θα δωθεί στον p1 (b και p2 περιττοί).

Class RadioContestPosition:

Κάνει extend την κλάση BothPlayerDicePosition. Δεν προστίθεται κάποιο καινούριο attribute. Ακολουθούν οι μέθοδοι της :

public RadioContestPosition(int boardNumber, String day,int money)

· Constructor της κλάσης.

@Override

public void performAction(Board b, Player p1, Player p2)

· Εδώ, αρκεί να καλέσουμε την performAction της Base Class, BothPlayerDicePosition.

Class LoterryPosition:

Κάνει επίσης extend την κλάση BothPlayerDicePosition. Δεν προστίθεται κάποιο καινούριο attribute. Ακολουθούν οι μέθοδοι της :

public LoterryPosition(int boardNumber, String day,int money)

· Constructor της κλάσης.

@Override

public void performAction(Board b, Player p1, Player p2)

· Εδώ, χρειάζεται να δημιουργήσουμε ένα Input Dialog για τους 2 παίχτες μας και να χρησιμοποιήσουμε ένα ζάρι (δεν έχει σημασία ποιανού) προκειμένου να βρούμε ποιος θα είναι ο νικητής. Στην συνέχεια τον ανακοινώνουμε και με τις μεθόδους που έχουμε δει ως τώρα του καταθέτουμε το χρηματικό έπαθλο.

Πακέτο Dice

Σχετικά μικρό πακέτο, περιλαμβάνει την κλάση Dice η οποία υλοποιεί ένα ζάρι, που το έχουμε δει ήδη αρκετές φορές να χρησιμοποιείται σε προηγούμενα πακέτα και κλάσεις.

Class Dice:

Κάνει implement την Serializable.

Attributes:

private int lastdice; // πεδίο που κρατά την τελευταία ζαριά που έγινε.

Μέθοδοι:

public int roll()

· Παράγουμε έναν τυχαίο αριθμό μεταξύ 1-6 τον αναθέτουμε στο πεδίο lastdice και έπειτα τον επιστρέφουμε.

public int getLastdice()

· Accessor και συγκεκριμένα Getter, επιστρέφει το πεδίο lastdice.

Πακέτο Tile

Επίσης σχετικά μικρό πακέτο, περιέχει μόνο την κλάση Tile η οποία υλοποιεί την ιδέα του πλακιδίου ενός ταμπλού. Θα μπορούσε η ίδια η position με μερικά extra attributes να κάνει κάτι τέτοιο, ωστόσο θεώρησα πως δίνει μια πιο καθαρή υλοποίηση το να κάνω μία ξεχωριστή κλάση.

Class Tile:

Κάνει implement την Serializable.

Attributes:

private String image;	//Το όνομα της φωτογραφίας του πλακιδίου π.χ. //yardsale.png
private Position position;	//Η τοποθεσία του πλακιδίου π.χ. βρίσκεται στην ημέρα //12 - Παρασκευή
public int x,y;	//Συντεταγμένες που θα χρειαστούμε αργότερα στην κλάση //View για να τυπώνουμε ευκολότερα τους παίκτες στα //αντίστοιχα tiles.

Μέθοδοι:

public Tile(String image, Position position)

· Constructor της κλάσης.

public Position getPosition()

· Accessor και συγκεκριμένα Getter, επιστρέφει το πεδίο position.

public String getImage()

· Getter, επιστρέφει το πεδίο image της κλάσης.

public int getBoardNumber()

· Getter, επιστρέφει το πεδίο boardNumber από το πεδίο position της κλάσης μας.

public void action(Board board, Player p1, Player p2)

· Μέθοδος η οποία καλεί την μέθοδο performAction του πεδίου position της κλάσης μας. (Εδώ για παράδειγμα γίνεται εμφανές το πόσο βολικό είναι που δεν χρειάζεται να παίρνω περιπτώσεις για να περάσω ορίσματα απλώς για την κλήση της performAction.)

Πακέτο Board

Φτάσαμε, λοιπόν, στο τελευταίο πακέτο του πακέτου Model. Σε αυτό το πακέτο περιέχεται η ιδιαιτέρως σημαντική κλάση, Board που υλοποιεί την έννοια του ταμπλού του παιχνιδιού μας με ό,τι αυτό περιλαμβάνει. Θα την δούμε αναλυτικά παρακάτω.

Class Board:

Κάνει implement την Serializable.

Attributes:

```
private int months;          //Αριθμός μηνών που θα διαρκέσει το παιχνίδι.

private int jackpot;         //Τα χρήματα που βρίσκονται στο Jackpot.

Vector<Tile> tiles = new Vector<Tile>();

//Vector που περιέχει όλα τα πλακίδια του ταμπλού μας.

Vector<Card> mailCards = new Vector<Card>();

//Vector που περιέχει όλες τις κάρτες μηνυμάτων.

Vector<Card> dealCards = new Vector<Card>();

//Vector που περιέχει όλες τις κάρτες συμφωνίας

public ClassLoader cldr = this.getClass().getClassLoader();

// Βοηθητικό attribute για την μέθοδο readFile που θα δούμε σε λίγο.
```

Μέθοδοι:

```
public void readFile(String path, String type)
```

· Με βάση το δοθέντα μονοπάτι και τύπο η εν λόγω μέθοδος διαβάζει τις εικόνες, μηνύματα από τον φάκελο resources του project δημιουργεί στιγμιότυπα καρτών Mail/Deal και τις αποθηκέυει σε αυτά.

```
private void tiles_init()
```

· Μέθοδος με την οποία αρχικοποιούνται τα πλακίδια του ταμπλού μας. Συγκεκριμένα, τοποθετούμε στο Vector tiles το κατάλληλο πλήθος από κάθε είδος πλακιδίου αρχικοποιώντας το με τις κατάλληλες παραμέτρους. Το κάθε πλακίδιο γίνεται instantiate με την δικό του boardNumber ο οποίος δεν είναι κοινός για κανένα άλλο.

```
public void init_board()
```

· Μέθοδος με την οποία καλούμε την tiles_init και αρχικοποιούμε την τιμή του jackpot σε μηδέν.

public Board(int months)

· Constructor της κλάσης, καλεί την readFile για την δημιουργία των καρτών Deal και Mail, στην συνέχεια την init_board και τέλος αναθέτει τιμή στο πεδίο months της κλάσης.

public void setJackpot(int jackpot)

· Transformer αναθέτει τιμή στο πεδίο jackpot.

public int getJackpot()

· Accessor και συγκεκριμένα Getter, επιστρέφει το πεδίο jackpot.

public int getMonths()

· Accessor και συγκεκριμένα Getter, επιστρέφει το πεδίο months της κλάσης.

public Tile getTileByDay(int day)

· Επίσης Accessor και συγκεκριμένα Getter, επιστρέφει το πλακίδιο το οποίο εχεί τιμή ίσι με day (arg) στο πεδίο day του . Αν δωθεί μέρα που δεν αντιστοιχεί στο ταμπλό π.χ. 32 τότε επιστρέφει null.

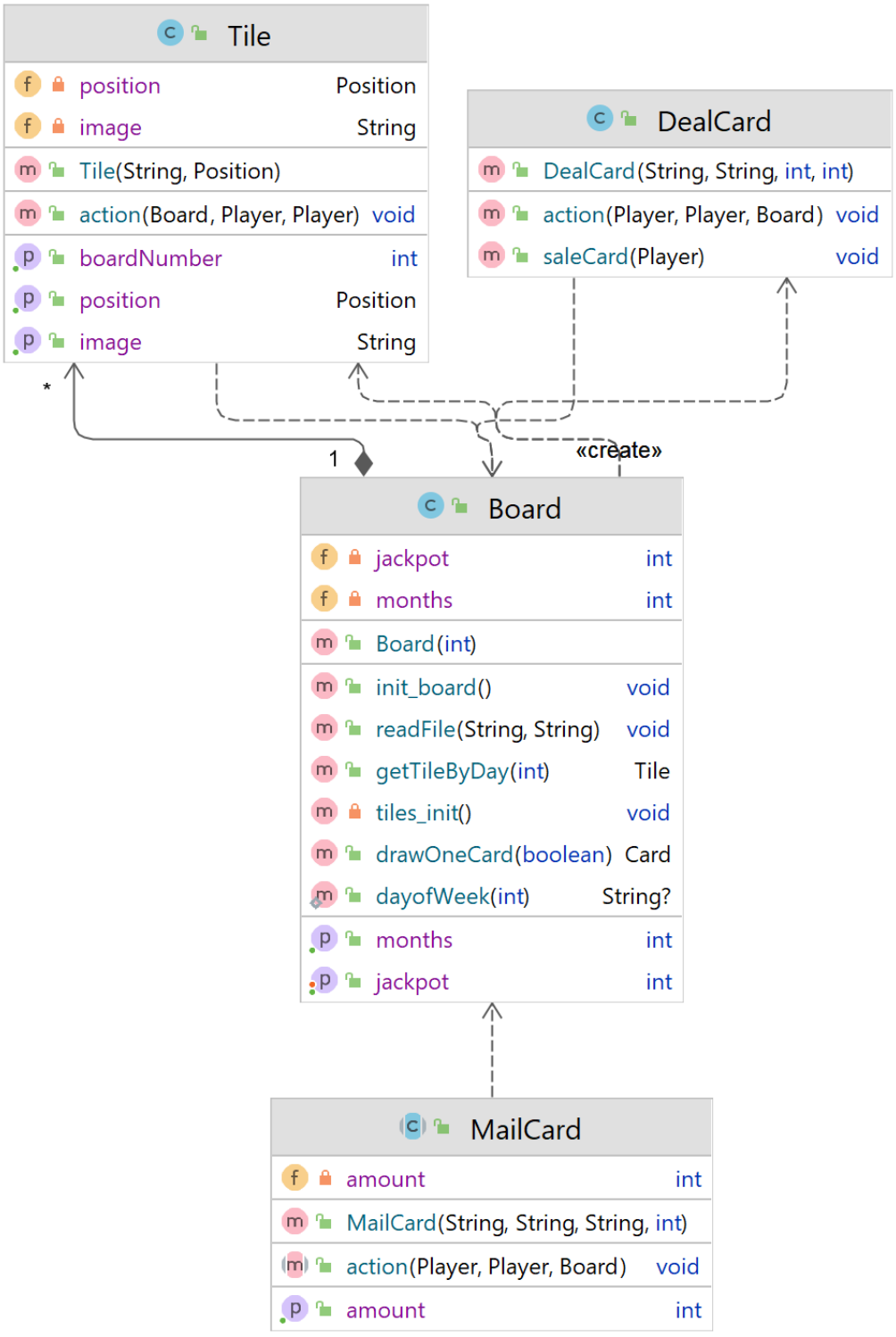
public Card drawOneCard(boolean choice)

· Transformer, αν το choice είναι true, τότε τραβάει μια τυχαία κάρτα συμφωνίας (Deal) την αφαιρεί και την επιστρέφει. Αν είναι false τότε το ίδιο, αλλά για μία κάρτα μηνύματος (Mail).

public static String dayofWeek(int dayOfMonth)

· Βοηθητική μέθοδος για να μπορούμε ευκολότερα να βρίσκουμε σε ποια μέρα της εβδομάδας αντιστοιχεί η εκάστοτε ημερομηνία/ημέρα του μήνα. Είναι δηλωμένη ως static ώστε να έχουμε πρόσβαση σε αυτή χωρίς να δημιουργήσουμε στιγμιότυπο της κλάσης, μιας και θα μας χρειαστεί και σε κλάσεις εκτός της Board.

Όλα αυτά τα χαρακτηριστικά που περιγράψαμε παραπάνω για την κλάση Board, μπορούμε να τα απεικονίσουμε και με ένα διάγραμμα UML :


Tile
position Position
image String
Tile(String, Position)
action(Board, Player, Player) void
boardNumber int
position Position
image String
DealCard
DealCard(String, String, int, int)
action(Player, Player, Board) void
saleCard(Player) void
*
1
«create»
Board
jackpot int
months int
Board(int)
init_board() void
readFile(String, String) void
getTileByDay(int) Tile
tiles_init() void
drawOneCard(boolean) Card
dayofWeek(int) String?
months int
jackpot int
MailCard
amount int
MailCard(String, String, String, int)
action(Player, Player, Board) void
amount int

3. Η Σχεδίαση και οι Κλάσεις του Πακέτου Controller

Class Controller:

Ο τρόπος με τον οποίο θα αλληλοεπιδρά ο controller με το model και το view είναι σαφής. Η κλάση Controller θα περιέχει ως attributes τους βασικούς πυλώνες του προγράμματος, δηλαδή ταμπλό παίκτες και γραφική απεικόνιση. Έτσι, με τις μεθόδους που ορίσαμε (και θα ορίσουμε για το πακέτο View) θα μπορούμε να λαμβάνουμε, αλλά και να στέλνουμε πληροφορίες σύμφωνα με την εξέλιξη του παιχνιδιού.

Attributes:

```
private Player p1, p2;          //Οι 2 παίκτες του παιχνιδιού μας
private Board board;            //Το ταμπλό του παιχνιδιού μας
private View view;              //Η γραφική απεικόνιση – GUI του παιχνιδιού μας
```

Μέθοδοι:

```
public void init()
```

· Η μέθοδος αυτή, σε περίπτωση που ξεκινάμε καινούρια παρτίδα, αρχικοποιεί το παιχνίδι μας προκειμένου να ξεκινήσει η ροή του παιχνιδιού. Εδώ κάνουμε instantiate τα attribute μας : p1, p2, board και view. Κάνουμε ορατό το παράθυρο της View, χρησιμοποιούμε την μέθοδο printPlayer της View για να τυπώσουμε στο ταμπλό τους παίκτες μας (τις μεθόδους της View θα τις δούμε στην παραπάνω ενότητα).Ανακοινώνουμε το ποιος θα παίξει πρώτος και στην συνέχεια όσο το παιχνίδι δεν έχει τελειώσει, καλούμε την μέθοδο turn για να ξεκινήσει ο γύρος του παίχτη. Μόλις τελειώνει ο γύρος του, ξανά καλούμε την turn με ορισμα τον παίκτη που δεν έπαιξε πριν και ούτω καθεξής.

```
public void loadstate(Object p1, Object p2, Object board, Object view)
```

· Σε περίπτωση που χρειαστεί να φωρτώσουμε αποθηκευμένη παρτίδα παιχνιδιού, η εν λόγω μέθοδος αναλβάνει να εκχωρήσει στα πεδία της κλάσης Control τα αντικείμενα που είχαν φτιαχτεί στην παλιά παρτίδα και τέλος να κάνει visible το frame. Είναι δηλαδή σαν την μέθοδο init, αλλά δεν αφορά την δημιουργία καινούριας παρτίδας.

```
public int loadturn()
```

· Όταν φορτώνουμε μια παλιά παρτίδα, μια ακόμα σημαντική πληροφορία εκτός από την κατάσταση στην οποία βρισκόταν τα πεδία του Controller είναι η σειρά των παικτών με την οποία θα συνεχιστεί το παιχνίδι. Έτσι πρέπει να δούμε ποιος παίκτης πάτησε το save ώστε να είναι αυτός που θα ξεκινήσει. Αυτό το επιτυγχάνουμε με αυτή τη μέθοδο, η οποία

διαβάζει το data από τον υπολογιστή του χρήστη που αφορά την σειρά (order.txt) και επιστρέφει τον αριθμό αυτού του παίκτη προκειμένου να το διαχειριστεί στη συνέχεια η μέθοδος start().

public void start(boolean save)

· Τα βήματα που πρέπει να ακολουθήσουμε είτε ξεκινήσουμε νέα, είτε φορτώσουμε παλιά παρτίδα, μετά από ένα σημείο είναι κοινά. Αυτά τα κοινά βήματα, λοιπόν, υλοποιεί αυτή εδώ η μέθοδος. Συγκεκριμένα, βάση του ορίσματος save(true αν θέλουμε παλιά παρτίδα, false για καινούρια) αποφασίζει αν πρέπει να καλέσει την μέθοδο loadturn ή την init και στην συνέχεια διατηρεί την σειρά με την οποία οι παίκτες παίζουν, ελέγχει με την GameFinished() αν το παιχνίδι τελείωσε και αν τελείωσε καλεί την printwinner της κλάσης view και κάνει exit το παιχνίδι.

public int whoPlaysFirst()

· Μέθοδος η οποία καθορίζει ποιος παίκτης θα ξεκινήσει πρώτος, επιστρέφοντας τυχαία τον αριθμό του. (π.χ. 1 ή 2)

public boolean GameFinished()

· Observer, επιστρέφει true αν το παιχνίδι έχει τελειώσει, διαφορετικά false. Το σκεπτικό πίσω από τον υπολογισμό της κατάστασης στην οποία βρίσκεται το παιχνίδι, δηλαδή αν τελείωσε ή συνεχίζεται ακόμα, είναι : Για να έχει τελειώσει το παιχνίδι θα πρέπει ο αριθμός μηνών του Board να είναι ίσος με τον αριθμό των paydays του κάθε παίκτη. Επομένως μας είναι ιδιαίτερα χρήσιμες οι μέθοδοι : getMonths() και getNumOfPaydays().

private Player getOtherPlayer(Player p)

· Βοηθητική μέθοδος για να μπορούμε ευκολότερα/γρηγορότερα να περνάμε ως όρισμα τον άλλον παίχτη του παιχνιδιού σε κάποια μέθοδο (π.χ. στην drawOneCard). Είχαμε αναλύση σε προηγούμενες ενότητες την σημασία των μεθόδων ενεργειών των καρτών/θέσεων (π.χ. performAction) να έχουν όμοιες υπογραφές. Επειδή δεν γνωρίζουμε εκ των προτέρων πότε μία ενέργεια χρειάζεται και τους 2 παίκτες είναι σημαντικό να περνάμε τον current παίχτη που παίζει τώρα/είναι η σειρά του, ως argument για το p1 και τον παίχτη που δεν παίζει ως p2. Επειδή όμως στην μέθοδο turn που θα δούμε πιο κάτω έχουμε ως όρισμα το p και είναι κοινή μέθοδος και για τους 2 παίκτες, μπορούμε εύκολα να κάνουμε την εναλλαγή στα ορίσματα στις μεθόδους action, με την σωστή τοποθέτησή τους μέσω της getOtherPlayer.

public int getScore(Player p)

· Observer, επιστρέφει το score του παίχτη, το οποίο υπολογίζεται μέσω της διαφοράς του πορτοφολιού του παίχτη μείον τα δάνεια και τον φόρο που πληρώνει για αυτά. Χρήσιμοι μέθοδοι για αυτόν τον υπολογισμό είναι : getWallet()/getDebts().

public int turn(Player p)

· Μέθοδος ιδιαίτερης σημασίας. Υλοποιεί την σειρά του παίχτη στο παιχνίδι με ό,τι αυτό συνεπάγεται. Δηλαδή, μέσω αυτής ο παίχτης μπορεί να ρίξει ζάρια, να μετακινηθεί, να

τραβήξει κάρτες, να τις πουλήσει κλπ. Περιλαμβάνει στο εσωτερικό της κλάση turn που κάνει Implement την κλάση Runnable, προκειμένου να τρέχουμε νήματα και να δίνουμε χρόνο στον παίχτη να πατήσει το εκάστοτε κουμπί στο GUI όποτε νιώθει έτοιμος. Όπως, φυσικά, μπορείτε να καταλάβετε, η μέθοδος turn είναι η μεγαλύτερη σε έκταση εώς τώρα, και όχι αδίκως.

public void giveLoan(Player p)

· Μέθοδος η οποία καλείται από την turn, όταν ο παίκτης πατήσει το κουμπί “Get Loan” προκειμένου να πάρει δάνειο. Αρχικά εμφανίζει Input Dialog για την επιλογή του ποσού δανείου και στην συνέχεια η συναλλαγή πραγματοποιείται μέσω των : setDebts/setWallet/ getDebts/getWallet.

public void movePlayer(Player p,int diceNum)

· Μέθοδος με την οποία βάσει της ζαριάς του παίχτη, υπολογίζουμε την νέα του τοποθεσία στο ταμπλό και στην συνέχεια μέσω της printPlayer της κλάσης View το παρουσιάζουμε και στην γραφική απεικόνιση. Η μέθοδος είναι δομημένη έτσι ώστε να μην μπορεί να παραληφθεί η ημέρα PayDay.

4. Η Σχεδίαση και οι Κλάσεις του Πακέτου View

Το πακέτο view, που είναι και το τελευταίο πακέτο του προγράμματός μας, αποτελείται από 4 κλάσεις, ορισμένες εκ των οποίων έχουν αναφερθεί και σε προηγούμενες ενότητες. Εδώ πρόκειται να τις δούμε αναλυτικά. Συγκεκριμένα, αυτό το πακέτο περιλαμβάνει την κλάση View, που υλοποιεί την γραφική απεικόνιση του παιχνιδιού μας, την DiceWindow η οποία ανοίγει νέο παράθυρο προκειμένου ένας ή και οι 2 παίκτες να ρίξουν ζάρια λόγω κάποιου tile, SellCardWindow, η οπόια ανοίγει παράθυρο προκειμένου ο χρήστης να επιλέξει ποια από τις κάρτες συμφωνίας που έχει αγοράσει θέλει να πουλήσει (ανοίγει όταν πέσει σε positon Buyer) και τέλος η κλάση ViewDealCards που μοιάζει με την SellCardWindow, μόνο που ο ρόλος της είναι απλώς να παρουσιάσει τις κάρτες συμφωνίας του παίχτη και όχι να γίνει κάποια επιλογή για πώληση.

Class View:

Η κλάση View κάνει extend την κλάση JFrame και implements την κλάση ActionListener, Serializable. Αυτό γίνεται για να μπορούμε να υλοποιήσουμε το GUI, καθώς και για να δώσουμε χρησιμότητα στα κουμπιά της εφαρμογής (βλέπε public void actionPerformed(ActionEvent ae)).

Attributes:

```
    JLayeredPane layeredPane;

//Μπορουμε να το σκεφτούμε σαν τον καμβά του παραθύρου. Εκεί θα stackάρουμε τα περισσότερα JLabels.

    JTextArea infobox;

// Το κουτί στο οποίο θα παρουσιάζονται πληρφορίες του παιχνιδιού όπως πόσοι μήνες απομένουν ακόμα.


    JLabel background;          //Το background του παραθύρου

    JLabel player1box;          //Το κουτί του 1ου παίχτη

    JLabel player2box;          //Το κουτί του 2ου παίχτη

    JButton rollDice1, rollDice2;          //Τα κουμπιά για τα ζάρια των 2 παικτών

    JButton mydealCards1, mydealCards2;    //Τα κουμπιά "My Deal Cards" των 2
                                           //παικτών

    JButton getLoan1, getLoan2;            //Τα κουμπιά για δάνειο των 2 παικτών

    JButton endTurn1, endTurn2;            //Τα κουμπιά για την ολοκλήρωση του γύρου του
                                           //κάθε παίκτη

    JButton move1, move2;                  //Τα κουμπιά για την μετακίνηση των παικτών
                                           //στο ταμπλό

    JLabel dice1, dice2;                   //Η φωτογραφία των ζαριών των 2 παικτών

    JButton mailCard,dealCard;             //Κουμπιά για κάρτες Μηνυμάτων/Συμφωνίας

    JLabel banner;                         //Η φωτογραφία του banner του παιχνιδιού
                                           //PAYDAY

    JLabel jackpot;                        //Η φωτογραφία του jackpot

    JLabel jackpotText;                    //Κείμενο με το status του jackpot

    JLabel pawn1,pawn2;                    //Φωτογραφίες για τα 2 πιόνια

    public int action;                     //μεταβλητή που αποτελεί γέφυρα μεταξύ της
                                           //κλάσης View-Controller.Ενημερώνει τον
                                           //controller για το ποιο κουμπί πατήθηκε

    private Board b;                       //Το ταμπλό που θα κάνουμε display
```

Μέθοδοι:

static public int selectDialog()

· Μέθοδος η οποία ανοίγει dialog Box στην αρχή της εφαρμογής και δίνει την επιλογή στον χρήστη να διαλέξει αν θέλει να ξεκινήσει καινούρια παρτίδα ή να φορτώσει μια παλιά(την οποία προφανώς είχε αποθηκεύσει όταν έπαιζε προηγουμένως με το κουμπί SAVEDATA).Η μέθοδος επιστρέφει την επιλογή του χρήστη προκειμένου στην συνέχεια να γίνει κατάλληλο handle αυτής.

static public int durationDialog()

· Μέθοδος η οποία ανοίγει dialog Box όταν ξεκινάει νέα παρτίδα και δίνει την επιλογή στον χρήστη να διαλέξει την διάρκεια της παρτίδας σε μήνες. Έπειτα επιστρέφει την επιλογή του χρήστη, προκειμένου να περαστεί ως όρισμα στο αντικείμενο board του παιχνιδιού. Είναι σημαντικό η μέθοδος αυτή να υλοποιηθεί με τέτοιον τρόπο, ώστε να μην επιτρέπονται παράνομες τιμές όπως μήνες μεγαλύτεροι του 3 ή αρνητικοί. Επομένως πρέπει να γίνεται κατάλληλο handle ώστε να ξαναζητιέται έγκυρη τιμή από τον χρήστη σε περίπτωση λάθους.

public void printphotos()

· Μέθοδος με την οποία τυπώνουμε στο frame μας το background του παιχνιδιού, το banner και το jackpot.

public void printinfobox()

· Μέθοδος με την οποία τυπώνουμε στο frame μας το πλαίσιο του infobox (και το αντίστοιχο text)

public void printCardButtons()

· Μέθοδος με την οποία τυπώνουμε στο frame μας τα κουμπιά για να τραβήξουμε κάρτες μηνύματος/συμφωνίας. Δεν ξεχναμε να κανουμε addActionListener.

public void printPlayerBoxes(Board board, Player p1, Player p2)

· Μέθοδος με την οποία τυπώνουμε στο frame μας τα πλαίσια των 2 παικτών μας, μαζί με τα κουμπιά που περιλαμβάνει το καθένα, και τις πληροφορίες τους π.χ. τα χρήματα του πορτοφολιού τους κλπ. (βάζουμε actionListener στο κάθε κουμπι).

public void printTiles(Board board)

· Μέθοδος με την οποία τυπώνουμε στο frame μας όλα τα tiles του ταμπλού μας τα οποία παίρνουμε από την παράμετρο board, χρησιμοποιώντας την μέθοδο getTileByDay και την getImage. Αξίζει να σημειωθεί εδώ, πως σε αυτή τη μέθοδο αξιοποιούνται τα πεδία x,y της κλάσης Tile. Συγκεκριμενα αφου ορίσουμε την θέση του πρώτου tile στο frame

μας αρχίζουμε με προφανή τρόπο να υπολογίζουμε και την τοποθεσία στο frame των υπολοίπων. Μαζί με την τοποθέτησή τους, αποθηκεύουμε τις συντεταγμένες που χρησιμοποιήσαμε στα αντίστοιχα attributes x,y, ώστε στην συνέχεια να μπορούμε να εκτυπώσουμε ευκολότερα τα πιόνια στο κάθε tile του ταμπλού μας.

public void printwinner(int num)

· Σχετικά μικρή και απλή μέθοδος, η οποία με βάση τον αριθμό παίκτη που έλαβε ως όρισμα, ανοίγει dialog Box και τον παρουσιάζει ως νικητή του παιχνιδιού. Προκειται για μέθοδο που καλείται από την κλάση Controller.

public void ThursdaySunday(Player p)

· Μέθοδος που αφορά τις περιπτώσεις : «Ποδοσφαιρικός Αγώνας Κυριακής» και «Άνοδος αξίας κρυπτονομισμάτων» . Συγκεκριμένα, με βάση την τοποθεσία του παίκτη στο ταμπλό, συμπεραίνουμε αν βρίσκεται στην ημέρα Πέμπτη ή Κυριακή και του πετάμε αντίστοιχο dialog box για να επιλέξει αν θέλει να κάνει πρόβλεψη. Στη συνέχεια με βάση την απάντηση και το εσωτερικό ρολάρισμα των ζαριών (που ο παίχτης δεν ελέγχει/βλέπει) ανακοινώνουμε αν η πρόβλεψη του ήταν σωστή ή όχι και του δίνουμε/παίρνουμε τα αντίστοιχα χρήματα.

public void printDice(int playerNumber,int diceNumber)

· Μέθοδος με την οποία εκτυπώνουμε τον αριθμό ζαριάς σύμφωνα με το diceNumber στο ζάρι του παίκτη με αριθμό playerNumber.

public void printPlayer(Player p,Board board,int day)

· Μέθοδος με την οποία εκτυπώνουμε το πιόνι του παίκτη. Κάνουμε instantiate το πιόνι μόνο στην ημέρα 0, δηλαδή στο σημείο αφετηρίας ενώ τις υπόλοιπες ημέρες απλώς το μετακινούμε. Για την τύπωση στο εκάστοτε πλακίδιο του ταμπλού, χρησιμοποιούμε τα πεδία x,y που είδαμε παραπάνω από το board.Η ημέρα στην οποία πρέπει να εκτυπώσουμε το πιόνι εξαρτάται από την παράμετρο day.

public View(Board board,Player p1, Player p2)

· Constructor της κλάσης View. Σετάρει κάποια βασικά πράγματα για το frame μας, θέτει τιμή στο πεδίο board της κλάσης, κάνει Instantiate τον καμβά (layeredpane) και τα ζάρια του παιχνιδιού και στην συνέχεια καλεί τις μεθόδους : printphotos, printinfobox, printCardButtons, printPLayerBoxes και printTiles που είδαμε παραπάνω.

public void updateinfo(Board b, Player p1, Player p2)

· Μέθοδος με την οποία ενημερώνουμε τις πληροφορίες/status που κάνουμε display για τους παίκτες και για το jackpot. Χρησιμοποιείται από την κλάση Controller στην μέθοδο turn προκειμένου να κρατάει up to date στοιχεία κατά την διάρκεια του παιχνιδιού.

@Override

public void actionPerformed(ActionEvent ae)

· Μέθοδος με την οποία παρακολουθούμε πότε ένα κουμπί πατηθεί. Κάθε κουμπί, έχει έναν δικό του ακέραιο αριθμό, ο οποίος ανατίθεται στο πεδίο action της κλάσης μας. Έτσι η κλάση Controller παρακολουθεί το πεδίο action της κλάσης View προκειμένου να κάνει τις επιθυμητές ενέργειες σύμφωνα με το κουμπί που πατήθηκε.

Σημείωση : Τόσο τα πεδία x,y της Board, όσο και το πεδίο action της View θα μπορούσε να εφαρμοστεί encapsulation, ωστόσο λόγω χρόνου και βολικότητας, αυτό παραλήφθηκε.

Class DiceWindow:

Η κλάση DiceWindow δημιουργεί ένα παράθυρο (Dialog) για τον κάθε παίκτη προκειμένου να ρίξει τα ζάρια όταν του ζητηθεί. Μόλις πατήσει το κουμπί ROLL μπορεί να αποχωρήσει με το κουμπί EXIT και το αποτέλεσμα της ζαριάς του αποθηκεύεται.

Attributes:

```
private static Dialog d;        //Το dialog της κλάσης
private int dicenum;            //Ο αριθμός που έφερε το ζάρι του παίκτη
private int player;             //Ο αριθμός του παίκτη που παίζει π.χ. 1 ή 2
```

Μέθοδοι:

```
public int getDicenum()
```

· Accessor συγκεκριμένα getter, επιστρέφει την ζαριά του παίκτη.

```
public DiceWindow(int playerNumber)
```

· Constructor της κλάσης, όλες οι απαραίτητες ενέργειες γίνονται εδώ. Δημιουργούμε νέο frame, φτιάχνουμε τα απαραίτητα κουμπιά, υλοποιούμε στο εσωτερικό συναρτήσεις actionPerformed(ActionEvent e) για εκείνα και εννοείται δημιουργούμε και ένα ζάρι με την βοήθεια της κλάσης Dice από το πακέτο model.

Class SellCardWindow:

Η κλάση SellCardWindow έχει τα εξής attributes :

private static Dialog d; // Το dialog της κλάσης

private int choice; // Εδώ αποθηκεύεται η επιλογή που έκανε ο παίκτης

Μέθοδοι:

public int getChoice()

· Accessor συγκεκριμένα getter, επιστρέφει την επιλογή που έκανε ο παίκτης.

public SellCardWindow(Player p)

· Constructor της κλάσης, δημιουργεί ένα frame, μέσω της getCardsOwned της κλάσης Player καλεί την createbutton τόσες φορές, όσες είναι και το πλήθος των καρτών του παίκτη p.

private void createbutton(DealCard dc, int i)

· Βοηθητική μέθοδος για τον constructor της κλάσης, δημιουργεί ένα κουμπί με βάση την κάρτα συμφωνίας που λαμβάνει ώς όρισμα. Έπειτα στο εσωτερικό υλοποιείται και η μέθοδος actionPerformed(ActionEvent e) για να γνωρίζουμε ποιο κουμπί πατήθηκε. Επειδή ωστόσο ο αριθμός των κουμπιών που θα έχει το frame δεν είναι γνωστός εκ των προτέρων, περνάμε και την παράμετρο i, που συμβολίζει τον αριθμό κουμπιού που πρόκειται να κατασκευαστεί ώστε στην actionPerformed να θέσουμε το choice ίσο με το i.Στην συνέχεια κλείνει το παράθυρο(η actionPerformed).

Class ViewDealCards:

Πρόκειται για μία κλάση η οποία είναι πανομοιότυπη με την SellCardWindow, με την διαφορά πως αυτή δεν έχει πεδίο choice και εξυπηρετεί τον χρήστη μόνο στην προβολή των καρτών συμφωνίας που διαθέτει ο ίδιος.

Πακετο Main

Τελευταίο και πολύ μικρό πακέτο. Περιλαμβάνει μία μόνο κλάση, την Main και όπως μπορούμε να καταλάβουμε, είναι η κύρια μέθοδος του προγράμματός μας.

Class Main implements Serializable:

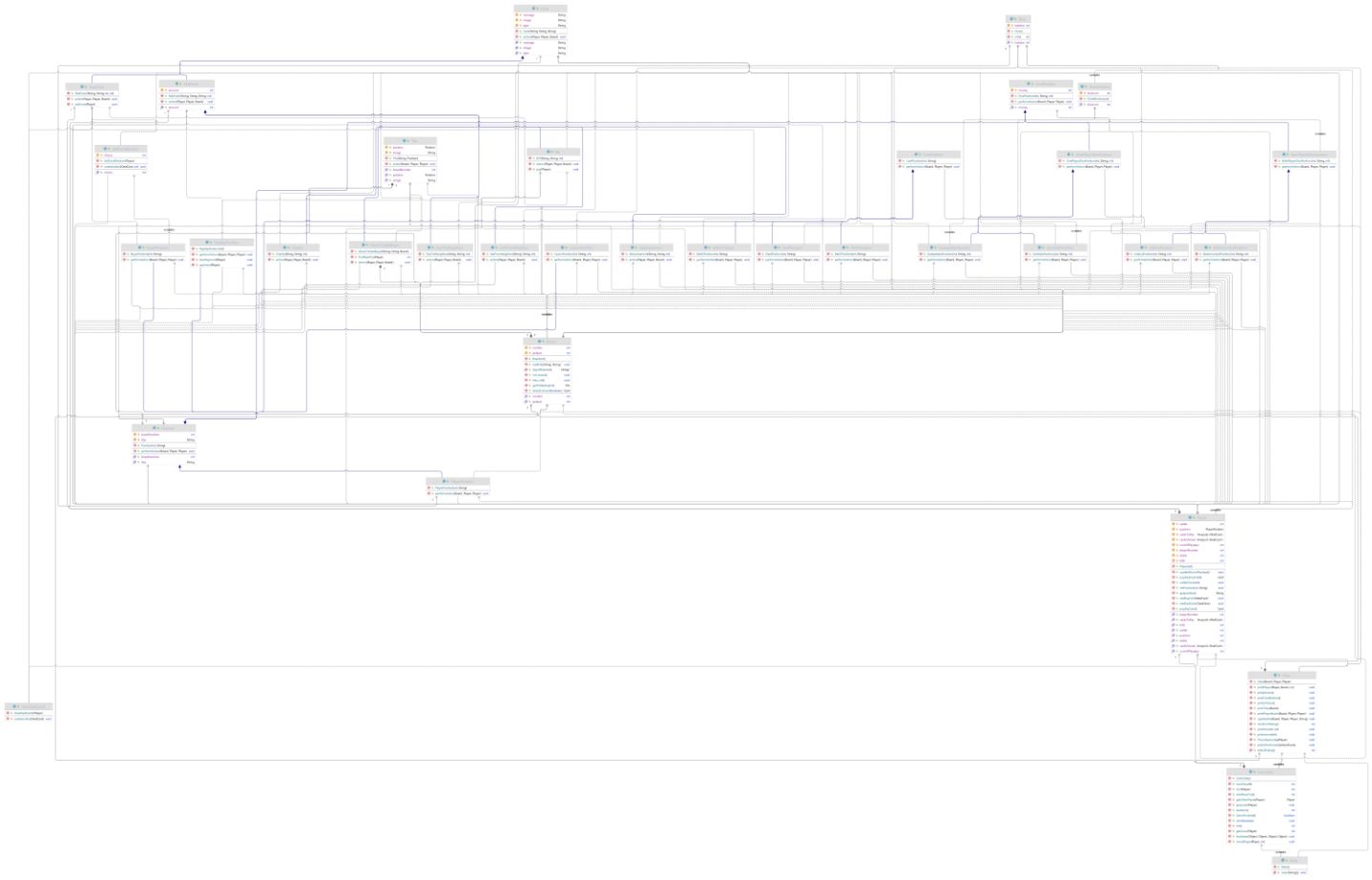
Εφόσον φτάσαμε στην τελευταία κλάση αυτού του project μπορούμε να μιλήσουμε για το implements Serializable το οποίο υπήρχε και σε προηγούμενες υπογραφές κλάσεων. Προκειμένου να μπορούμε να αποθηκεύουμε και να φορτώνουμε states του παιχνιδιού χρειάζεται να κάνουμε serialize κάποια objects(παίκτες, ταμπλό και παράθυρο). Προκειμένου να γίνει αυτό, οι κλάσεις των αντικειμένων μαζί με τα dependencies τους χρειάζεται να έχουν αυτήν την υπογραφή. Αξίζει να σημειωθεί πως χάρη στην κληρονομικότητα, η υπογραφή implements Serializable χρειάζεται μόνο σε parent classes.

Τώρα όσον αφορά την κλάση Main περιέχει μία μόνον μέθοδο :

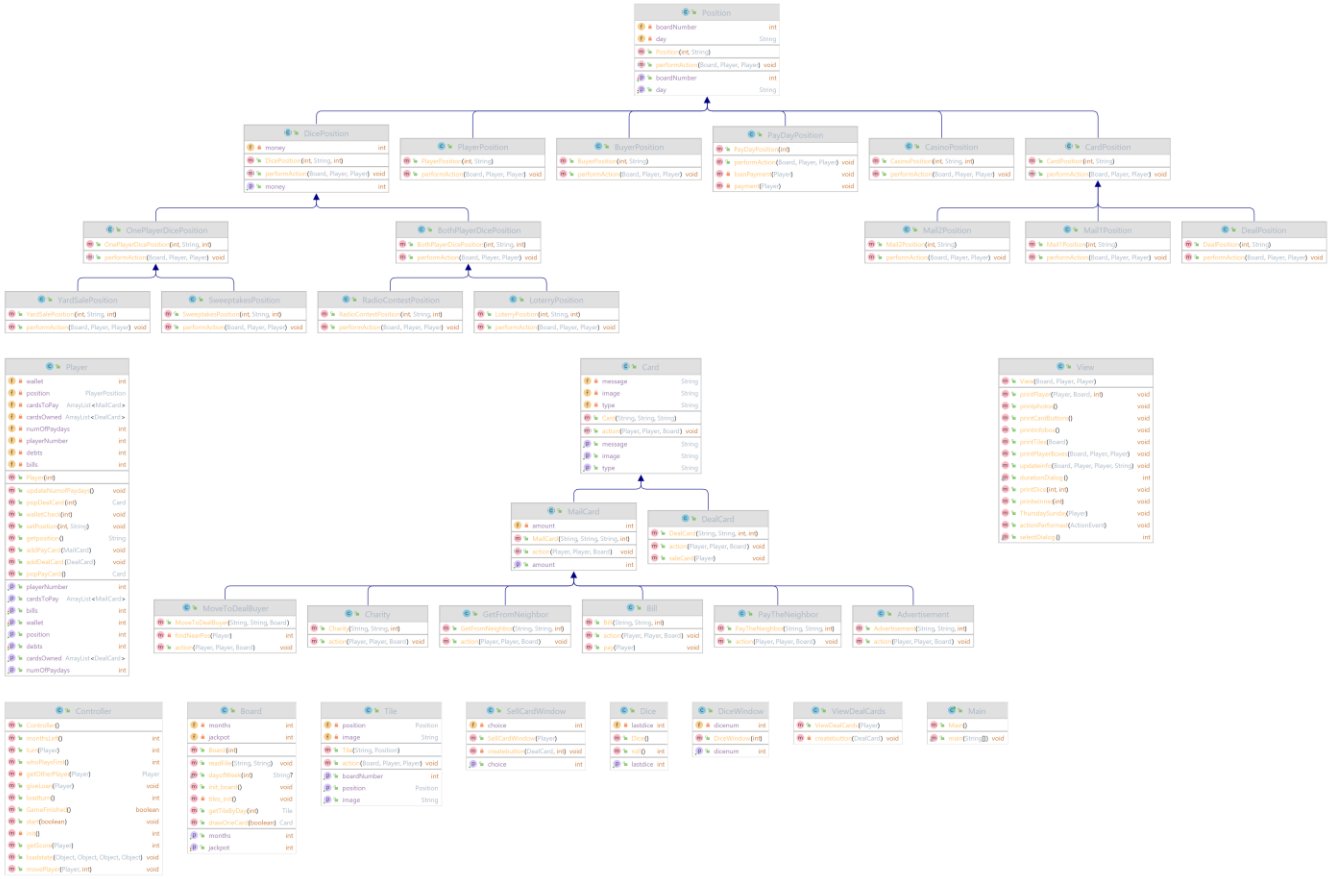
public static void main(String[] args)

· Η main μέθοδος της εφαρμογής μας. Δουλειά της είναι να καλεί την static μέθοδο selectDialog() της κλάσης View, προκειμένου να γνωρίζουμε αν θα ξεκινήσει καινούρια παρτίδα ή θα συνεχιστεί μια υπάρχουσα. Σε περίπτωση που ξεκινήσει καινούρια κάνουμε instantiate αντικείμενο της κλάσης Controller και καλούμε την μέθοδο του start() με παράμετρο false. Διαφορετικά, διαβάζουμε το αρχείο 'objects.txt' προκειμένου να φορτώσουμε τα παλιά πεδία του controller. Δημιουργούμε ένα νέο αντικείμενο Controller, καλούμε την μέθοδό του loadstate() με ορίσματα τα αντικείμενα που διαβάσαμε απ'το αρχείο προκειμένου να σεταριστούν και τέλος, καλούμε την μέθοδο start() με παράμετρο true.

5. Η Αλληλεπίδραση μεταξύ των κλάσεων – Διαγράμματα UML



Εδώ μπορούμε να παρατηρήσουμε όλες τις κλάσεις του προγράμματος μας με τα χαρακτηριστικά τους και τις σχέσεις που έχουν μεταξύ τους. Θα κάνουμε μια τελευταία ανακεφαλαίωση παρόλο που την δομή των κλάσεων την είδαμε παραπάνω με το ίδιο διάγραμμα χωρίς dependencies προκειμένου να γίνει ευκολότερα κατανοητό.



Πάνω έχουμε την κλάση Position με τα χαρακτηριστικά της. Μπορούμε τώρα να παρατηρήσουμε ότι από κάτω ακολουθούν οι κλάσεις που την κληρονομούν, όπως η DicePosition, η PlayerPosition κλπ. Βλέπουμε στην συνέχεια την Mail2Position, Mail1Position και DealPosition να κληρονομούν την CardPosition, στα δεξιά του διαγράμματος. Το ίδιο συμβαίνει και με την DicePosition που κληρονομείται από την OnePlayerDicePosition και από την BothPlayerDicePosition. Αυτές με την σειρά τους κληρονομούνται από τις YardSalePosition, SweeptakesPosition και RadioContestPosition,LotteryPosition αντίστοιχα .Στην συνέχεια μπορούμε να δούμε από κάτω την κλάση Card, η οποία κληρονομείται από την MailCard και DealCard. Έπειτα, κλάσεις όπως η Charity, Bill, κλπ κληρονομούν την MailCard. Στην συνέχεια μπορούμε να κοιτάξουμε την κλάση Player, η οποία περιέχει ως πεδίο MailCards και DealCards όπως επίσης και PlayerPosition. Στην συνέχεια μπορούμε να δούμε την κλάση Tile η οποία περιέχει πεδίο τύπου Position. Εξάρτηση συναντάμε και στην κλάση Board η οποία περιέχει εσωτερικά πεδίο τύπου Tile και Card. Η κλάση View με την σειρά της, χρειάζεται τις κλάσεις : ViewDealCards, DiceWindow και SellCardWindow. Τέλος η κλάση Controller περιέχει πεδία τύπου View, Player και Board, με αποτέλεσμα να ενώνει όλα τα παραπάνω. Κλείνοντας, απομένει η Main, η οποία χρειάζεται την κλάση Controller προκειμένου να δημιουργήσει ένα αντικείμενο τέτοιου τύπου και να εκτελέσει ορισμένες μεθόδους με αυτό.

6. Λειτουργικότητα (Β Φάση)

Όλα τα ερωτήματα της άσκησης, μαζί με το Bonus, υλοποιήθηκαν επιτυχώς. Λόγω χρόνου δεν πρόλαβα να κάνω generate test cases για όλες τις κλάσεις, παρά μόνο για : Board,Player,Position και Tile. Ωστόσο αξίζει να σημειωθεί πως το manual testing, παίζοντας δηλαδή το παιχνίδι με κάποιο μέλος της οικογένειας μου φάνηκε πολύ πιο βοηθητικό και διασκεδαστικό ☺ .

7. Συμπεράσματα

ΦΑΣΗ Α'

Μου ήταν αρκετά δύσκολο να σχεδιάσω την δομή του προγράμματος χωρίς να ξεκινήσω τις υλοποιήσεις των μεθόδων, οπότε μαζί με την σχεδίαση των κλάσεων έγινε και η υλοποίηση των μεθόδων. Αποδείχθηκε εν τέλει πως πολλά πράγματα που είχα στο μυαλό μου δεν μπορούσαν να υλοποιηθούν όπως τα φανταζόμουν ή δεν θα ήταν τόσο πρακτικά. Οπότε το να υλοποιώ τον κώδικα του προγράμματος με βοήθησε πολύ στο να κάνω ενέργειες που έχουν νόημα και να καταλήγουν σε κάποιο αποτέλεσμα. Επομένως το project ήδη από την 1η φάση είναι σε θέση να εκτελεστεί και να λειτουργήσει κανονικά μέχρι το τέλος, με εξαίρεση ελάχιστων λειτουργιών οι οποίες θέλουν διόρθωση.

ΦΑΣΗ Β'

Η φάση Β' του project επικεντρώθηκε κυρίως στο να διορθωθούν bugs που είχαν εντοπιστεί από την πρώτη κιόλας φάση και μερικών που προέκυψαν στην πορεία. Η δομή, λοιπόν δεν άλλαξε ιδιαιτέρως, με εξαίρεση προφανώς την προσθήκη μερικών ακόμα μεθόδων και την υλοποίηση της λειτουργίας της Αποθήκευσης του παιχνιδιού. Το μεγαλύτερο και δυσκολότερο μέρος της άσκησης υλοποιήθηκε στην Α' φάση, ενώ στην Β' ασχολήθηκα περισσότερο με λεπτομέρειες. Σε γενικές γραμμές, λοιπόν, η Β' φάση ήταν σχετικά ευκολότερη από την Α'.