Bilkent University

Department of Computer Engineering

# CS 319 Term Project

Design Report

*Bilpoly (Monopoly Bilkent)*

Section 3

Group 3H

**Project Group Members**

| | | |
|---|---|---|
| **1.** | Ömer Ünlüsoy | 21702136 |
| **2.** | Arda Akça Büyük | 21802835 |
| **3.** | İrem Tekin | 21803267 |
| **4.** | Ece Ünal | 21703149 |
| **5.** | Furkan Ahi | 21501903 |

**Supervisor:** Eray Tüzün

# 1. Introduction

## 1.1 Purpose of the System

Bilpoly is a computer game which can be played by two to four people on the same computer. It has two game modes: Bilkent Buildings Mode, Bilkent CS Mode and has two time modes: Normal Mode, Timed Mode. In Bilkent Buildings Mode the aim of each time mode is to dominate the whole Bilkent by opening Bilkas and Starbuckses to places and pushing the other players to go bankrupt. In Bilkent CS Mode, the gameplay is the same but instead of Bilkent buildings as lands there are CS courses and instead of money there is time that is measured with "hours". Players roll dice, move their pawns on the game board and try to win. The game's purpose is to provide a different version of the game Monopoly to make it exclusive to Bilkent, easy to navigate and play, fast and fun.

## 1.2 Design Goals

### 1.2.1 End User Criteria

#### 1.2.1.1 Usability

Any user that knows how to play the classic Monopoly game can learn how to play and navigate through the game in their first trial of the game, as long as we implement the project according to the rules that stated below to keep the game navigation and game play easy to learn, also to provide a pleasant game experience.

- At the game screen, with the game board, credit cards, stop button, roll dice button, next turn information, remaining time information and pop-ups that shows the recent player activities, the number of items does not exceed seven to make the gameplay less confusing. The players can also see whose turn it is from the credit card at the top from the credit card deck at the upper right. Also, to provide a pleasant game experience, they can see recent activities of the previous players from the pop-ups placed at the lower right bottom. Remaining time also will be displayed at the right part of the screen if the game is in the timed mode. The next turn information, which is placed at the right part of the screen, will be displayed to make the gameplay less confusing.

- At the main menu screen, there are only 4 buttons to make the user interface simple.

- At the player selection screen, there are two parts: one with the three buttons to choose player number and another one to choose player properties. In this part, panes that let the players enter their names and select pawns will be activated according to the number of players to avoid confusion.

- At the pre-game settings screen, there are three buttons to let the user decide the initial money, two buttons for board mode and two buttons for the time mode. All of them are labeled according to what they do to make the game self-explanatory. Also, there is a slider to let the user the time limit. When the cursor of the slider stops at some point, the user can see the time limit they choose in the label placed at the bottom of the slider.

- The background images are in lighter tones compared to other items on the screens to make the user experience more pleasant.

#### 1.2.1.2 Efficiency

The response time for the game is less than 0.5 seconds, the maximum amount that decided to not disrupt the game flow.

### 1.2.2 Developer/Maintainer Criteria

#### 1.2.2.1 Portability

The most important feature of the game developed is that it can be played on multiple platforms. Since this project is written in Java language and can be easily integrated into PC, MacOS and mobile platforms.

#### 1.2.2.2 Extendibility

The project can be improved according to the feedback from the user, developer and observers. Object-oriented class designs, interfaces and behaviors are designed in a way that will not affect the whole project in case the project is developed. As long as the new classes to be added do not change the gameplay, they enrich the project and create new modes and boards.

#### 1.2.2.3 Modifiability

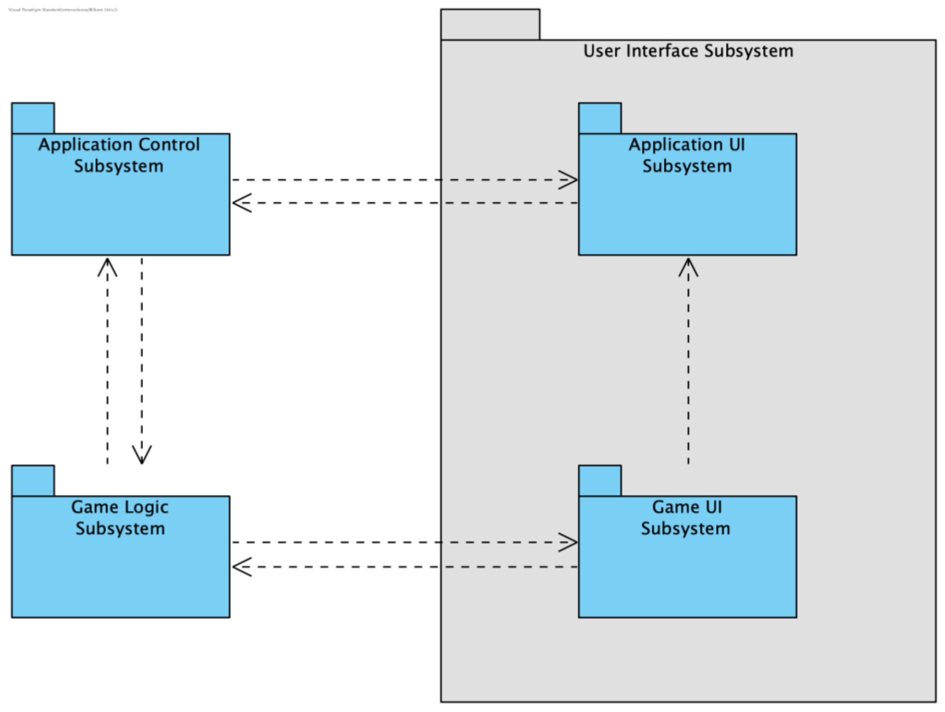
Writing the classes and interfaces directly allows the sub-parts of the project to be easily modified. Therefore, the change made on one side does not affect the parts of the other class unnecessarily. In addition, since a change made on the upper part will affect all lower parts, the implementation will be easier accordingly.

#### 1.2.2.4 Reusability

Since the general gameplay and rules of the game are largely fixed, changes made to the modes and boards do not have a great impact on the main code flow. Many of the new features that can be added will be able to be integrated into the game by reusing the source code and adapting it to the mode that will be designed with a few minor modifications.

# 2. High-level Software Architecture

## 2.1 Subsystem Decomposition



Bilpoly application has 3 main subsystems one of which consists of 2 subsystems;

**Application Control Subsystem**

- This subsystem has two main purposes, controlling the application with AssetManager and controlling the Main Menu elements with Application UI Subsystem.
- It includes control and entity objects of the application.
- It initializes the application and navigates players to create a game.
- It handles the screen changes in the Main Menu.
- It handles the sound effects, music, and background images.
- It creates a board and game elements to initialize a game.

**Game Logic Subsystem**

- This subsystem has two main purposes, controlling the game flow and controlling the game user interface with Game UI Subsystem.
- It includes control and entity objects of the application.
- This subsystem handles each turn during a game.
- It handles players and their properties as well as all the lands on the board.
- It handles all money flow and decides the winner at the end.
- This subsystem controls the game user interface by controlling the components of Game UI Subsystem.

**User Interface Subsystem**

- User Interface Subsystem handles Bilpoly application’s front-end (User Interface).
- It consists of two subsystems which control two separate user interface parts in the application.
- It includes boundary objects of the application.

1. **Application UI Subsystem**
    - Application UI Subsystem manages the User Interface of the Main Menu with its components; Options, Credits, How to Play, Player Selection, and Pre-Game Settings.
    - This user interface helps players initialize a game, open Credits and How to Play screens, etc.
    - User inputs via boundary objects of this subsystem manage the entity objects of Application Control Subsystem.

2. **Game UI Subsystem**
    - Game UI Subsystem manages the User Interface of the game screen, Pause Menu, and Game Over Screen.
    - This subsystem handles players' inputs during their turn.
    - It consists of the board, player deck, dice, history, and Pause Menu components.
    - User inputs via boundary objects of this subsystem manage the entity objects of Game Logic Subsystem.

## 2.2 Hardware/Software Mapping

Bilpoly game will be implemented in Java programming language and JavaFX libraries will be used for User Interface. For the execution of the application, Java Runtime Environment will be needed. Bilpoly application also needs at least Java Development Kit (JDK) 11 to be executed properly since this version supports JavaFX libraries.

Bilpoly game accepts game inputs via mouse rather than a keyboard since our way of implementation uses mostly buttons. However, users will need a keyboard to type their names to initialize a game. Moreover; some shortcuts for buttons can be implemented in the future, still in the process of consideration. Players also need a monitor to display the screen. Since the application will be lightweight, any computer with minimal hardware can be used.

Bilpoly can be run and played on macOS, Windows, and Linux systems with required software, Java Runtime Environment.

Players will not need any local or internet connection as Bilpoly can be played on one computer regardless of single or multiplayer game.

Database for data storage will not be needed since we will store any necessary data in local config files.

## 2.3 Persistent Data Management

Bilpoly application does not need a database system for data management. It will store data as config files, TXT files, and PNG or JPG files for images. Since Bilpoly does not need any previous data to initialize a game and will not modify these files, complex database system would be redundant. Since Bilpoly will not save any player object after the game, Bilpoly will not use JSON format. However, WAW, PNG, JPG, GIF formats will be used to store sound effects, music, and images.

## 2.4 Access Control and Security

Bilpoly will not use any kind of network connection or database system. Moreover, any personal or sensitive data will not be required or demanded rather than the players' names. It will not use any password protection, registration or login as it is not needed and anybody who downloaded the game can play Bilpoly. Most data will be deleted after each game.

Therefore, there will be no safety issues concerning the leakage of user passwords or sensitive data.

## 2.5 Boundary Conditions

### 2.5.1 Initialization

Bilpoly application will be started by opening the executable .jar file which will call the main method. The application will load the Main Menu with its components from the local filesystem. The background images and music will be loaded and initialized here. Players can upload new images and music via their local filesystem. After the Pre-Game Settings, the application will load the game screen components with a gameboard, player deck, etc.

Bilpoly will not use a database or local/internet network connection so it will not load any data from a database or make any request.

### 2.5.2 Termination

Bilpoly can be terminated by the “Exit” button on the Main Menu or the “Close” button of its window. If players decide to end the game and exit during a game; they can Pause the game, go to the Main Menu, and then push the “Exit” exit button to terminate the application.

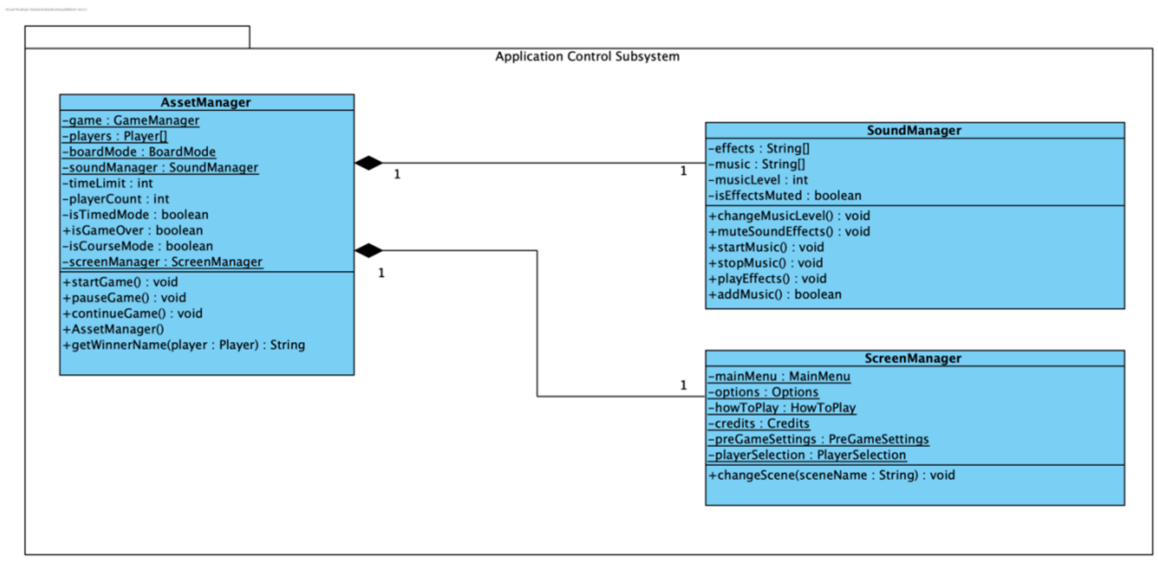
### 2.5.3 Failure

The game will terminate itself if it cannot find any required file in its local file path such as class files or UI images. However, the application will not terminate itself immediately if any sound effects, music, or background images files cannot be found. Rather the application will show a warning and players can close the application if they want.

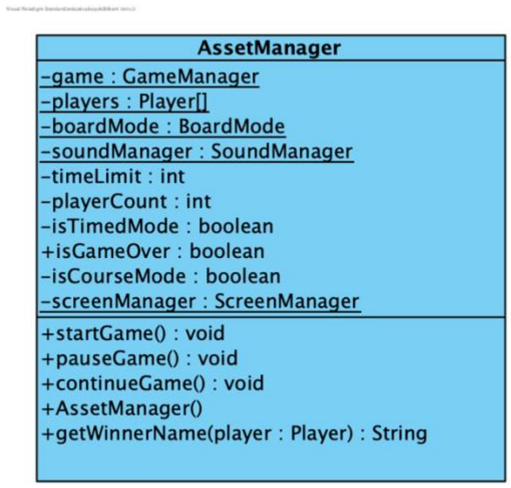
Application will terminate itself, if it crushes during a game. Since Bilpoly does not save any data, data loss is not the case.

# 3. Subsystem Services

## 3.1 Application Control Subsystem



### AssetManager Class



## Attributes

**private int timeLimit**: This attribute is to keep the time limit which is decided by the user in Timed Mode. This information comes from PreGameSettings UI class.

**private int playerCount**: This attribute is to keep the number of players which is decided by the user. This information comes from PlayerSelection UI class.

**private boolean isTimedMode**: This attribute holds the information whether the Time Mode is Normal Mode or Timed Mode. This information comes from PreGameSettings UI class.

**private boolean isCourseMode**: This attribute holds the information whether the Game Board Mode is Bilkent Buildings Mode or Bilkent CS Mode. This information comes from PreGameSettings UI class.

**private GameManager game**: This attribute is an instance of GameManager which initializes the GameManager with the information that comes from PreGameSettings and PlayerSelection UI classes.

**public boolean isGameOver**: This attribute holds the information about the state of the game (if it’s over or not). This information comes from GameManager which is a class in Game Logic Subsystem and the information is used in GameOver UI class.

**private SoundManager soundManager**: This attribute is an instance of SoundManager. Information comes from Options or PauseMenu UI classes.

**private Player player[]**: This attribute initializes players with the information comes from PlayerSelection UI class.

## Methods

**public void startGame()**: Takes information from PreGameSettings UI class and calls the function changeScene(...) which changes the scene to GameScreen UI, starts the timer in GameManager class. Also this method calls the startMusic().

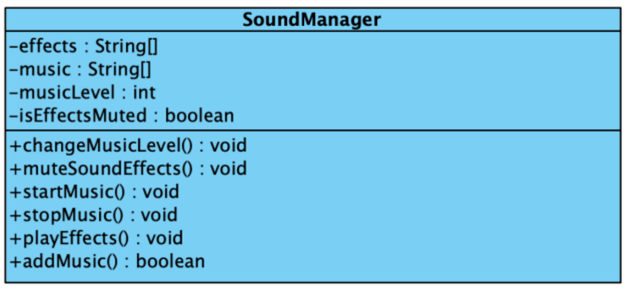
**public void pauseGame()**: Takes information from GameScreen UI class and stops the timer in GameManager class and calls the function changeScene(...) which changes the scene to PauseMenu UI.

**public void continueGame()**: Takes information from GameScreen UI class and resumes the timer in GameManager class and calls the function changeScene(...) which changes the scene to GameScreen UI.

**public AssetManager()**: The constructor for AssetManager.

**public String getWinnerName(player: Player)**: This method takes information from GameManager and  sends this information to GameOver UI class.

## SoundManager Class



## Attributes

**private  String effects[]:** This attribute holds the sound effect names as Strings.

**private String music[]**: This attribute holds the music names as Strings. This can be updated by addMusic(...) method.

**private int musicLevel:** This attribute holds the music level and can be updated by changeMusicLevel(...) method.

**private boolean isEffectsMuted:** This attribute holds the information whether sound effects are muted or not and this information can be updated by muteSoundEffects() method.

## Methods

**public void changeMusicLevel():** This method changes the music level according to the information that comes from Options UI and PauseMenu UI.

**public void addMusic(musicName: String):** This method updates the music[] with the information that comes from Options UI.

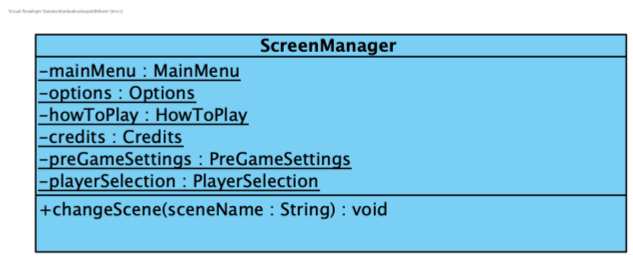
**public void muteSoundEffects()** : This method mutes the sound effects according to the information that comes from Options UI and PauseMenu UI.

**public void startMusic()**: This method starts the music.

**public void stopMusic()**: This method stops the music according to the information that comes from Options UI and PauseMenu UI.

**public void playEffects()**: This method plays the corresponding sound effects from the effects[] array.

## ScreenManager Class


ScreenManager
-mainMenu : MainMenu
-options : Options
-howToPlay : HowToPlay
-credits : Credits
-preGameSettings : PreGameSettings
-playerSelection : PlayerSelection
+changeScene(sceneName : String) : void

### Attributes

**private MainMenu mainMenu:** This attribute is an instance of MainMenu which initializes the mainMenu UI object.

**private Options options:** This attribute is an instance of Options which initializes the options UI object.

**private HowToPlay howToPlay:** This attribute is an instance of HowToPlay which initializes the howToPlay object.

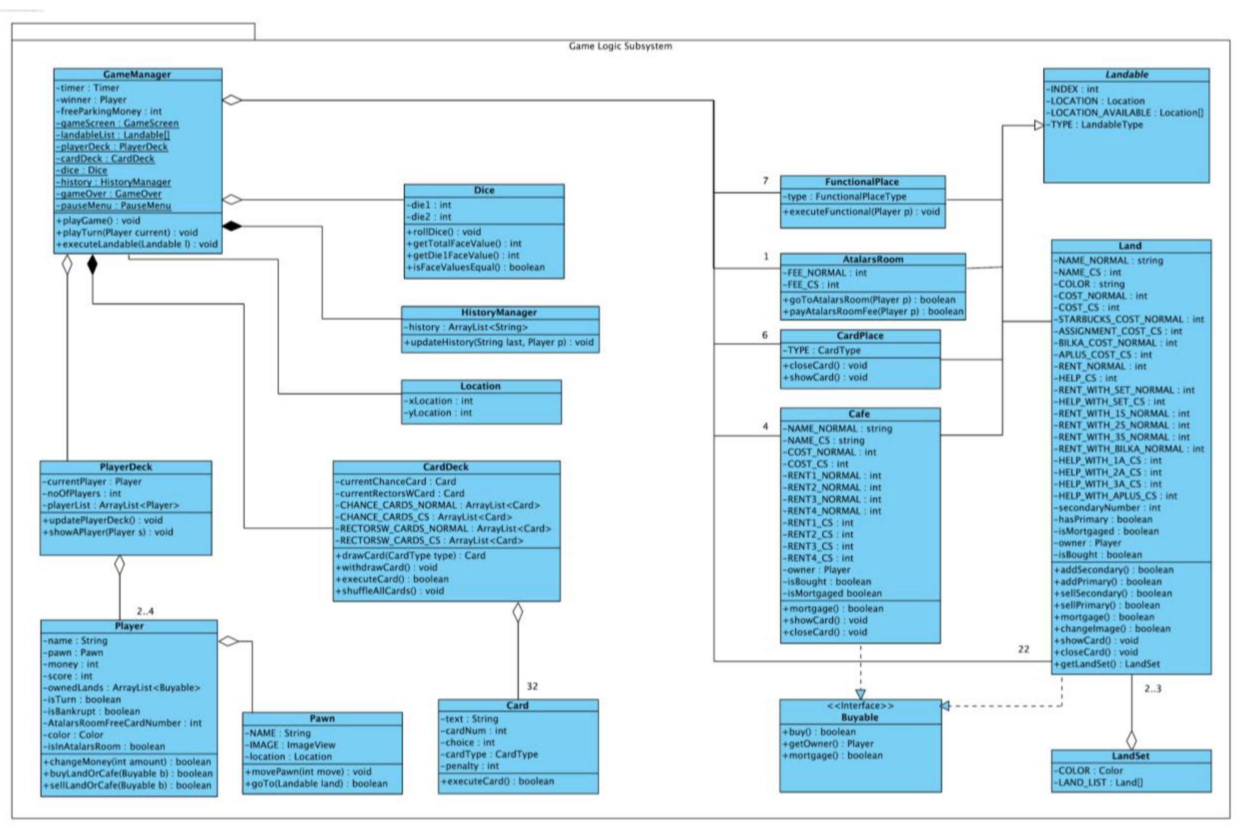
**private PreGameSettings preGameSettings:** This attribute is an instance of PreGameSettings which initializes the preGameSettings UI object.

**private PlayerSelection playerSelection:** This attribute is an instance of PlayerSelection which initializes the playerSelection UI object.

**Methods**

**public void changeScene(sceneName : String):** This method changes the scenes according to the information that comes from  the UI classes.

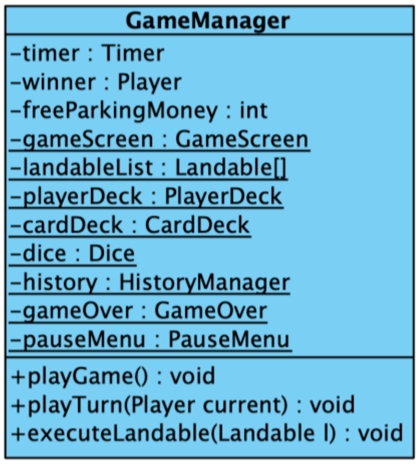
## 3.2 Game Logic Subsystem



Game Logic Subsystem consists of 15 classes, 1 abstract class, and 1 interface, and is responsible for the in-game flow of the software. It consists of real life Monopoly objects as classes and has functions needed to apply the game rules and game functionality.

The main class managing this subsystem is GameManager, which has the instances of every object and handles all of their operations. It has an instance of PlayerDeck, which holds all players inside it as an ArrayList and sorts it according to player turns. It has the list of Landables that differ in functionality as an array. It has the CardDeck instance which holds the Chance and Rector’s Whisper cards inside it. It has the Dice instance which has several functionalities through the game such as moving, getting out of jail, or picking the one who starts the game first. It has the HistoryManager in which the last three moves in the game are held. It also has the instances of GameOver and PauseMenu classes which are handled in the in-game UI.

## GameManager Class



### Attributes

**private Timer timer:** This attribute is the timer that counts down from the selected time limit for Timed Mode.

**private Winner winner:** This attribute is the Player object that wins the game which will be passed to the UI to display the winner.

**private int freeParkingMoney:** This attribute holds the free parking money. All money paid by the players to the bank (excluding the land sales) goes to the middle area. This attribute holds the money that is gone to the middle area. The player that lands on the “Free Parking” Landable takes all the money that piled up.

**private Landable[] landableList:** This attribute is an array of 40 Landables which are the each single cell of the board that the players can land. Landables differ in functionality. Some are buyable such as buildings, some are functional such as Atalar’s Room, and some make a player pick a card such as Rector’s Whisper Card.

**private PlayerDeck playerDeck:** This attribute is the instance of the PlayerDeck class which holds all players in an ArrayList. This instance is responsible for the operations on players (money changes, buying/mortgaging, going to Atalar’s Room etc.).

**private CardDeck cardDeck:** This attribute is the instance of the CardDeck class which holds all Chance and Rector’s Whisper Cards which all have different functionalities as arrays.

**private Dice dice:** This attribute is the instance of the class Dice which is used for selecting who starts the game and which is rolled each turn.

**private HistoryManager history:** This attribute is the instance of HistoryManager. HistoryManager holds the last 3 turns played in the game and shows them on the user interface. In each turn, history is updated.

**private GameOver gameOver:** This attribute is an instance of GameOver class which handles the GameOver screen (e.g. passing the winner) of the game.

**private PauseMenu pauseMenu:** This attribute is an instance of PauseMenu class which handles the pausing and resuming of the game.

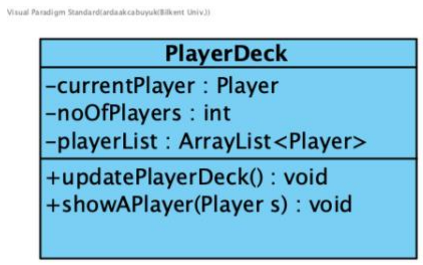
### Methods

**public void playGame():** This method manages the main game loop of the game. This method runs until the game ends.

**public void playTurn():** This method manages the player turn, which is called continuously from the playGame() method.

**public void executeLandable(Landable l):** This method executes the function of the Landable that the player has landed, according to the functionality of the Landable.

## PlayerDeck Class


Visual Paradigm Standard(ardaakcabuyuk(Bilkent Univ.))
PlayerDeck
-currentPlayer : Player
-noOfPlayers : int
-playerList : ArrayList<Player>
+updatePlayerDeck() : void
+showAPlayer(Player s) : void

### Attributes

**private Player currentPlayer:** This attribute is the Player object that has the turn.

**private int noOfPlayers:** This attribute holds the number of players in the game, which is initialized at the game setup stage.

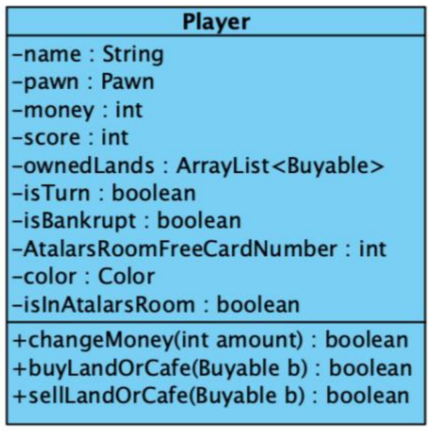
**private ArrayList<Player> playerList:** This attribute holds the players in the game, sorted according to player turns. This list will be used to sort the credit cards which represent the players.

### Methods

**public void updateDeck():** This function updates the deck when a player plays his/her turn. The function re-sorts *playerList* by moving the first player to the end.

**public void showAPlayer(Player s):** This function returns the information of the passed player, calls Game UI components to display selected player's info.

## Player Class


Player
-name : String
-pawn : Pawn
-money : int
-score : int
-ownedLands : ArrayList<Buyable>
-isTurn : boolean
-isBankrupt : boolean
-AtalarsRoomFreeCardNumber : int
-color : Color
-isInAtalarsRoom : boolean
+changeMoney(int amount) : boolean
+buyLandOrCafe(Buyable b) : boolean
+sellLandOrCafe(Buyable b) : boolean

### Attributes

**private String name:** This attribute is the name of the player.

**private Pawn pawn:** This attribute is the Pawn object that the player selects in game setup.

**private int money:** This attribute is the current balance of the player.

**private ArrayList<Buyable> ownedLands:** This attribute contains all the Buyables that the player has bought so far.

**private boolean isTurn:** This attribute is true only when this player has the turn.

**private boolean isBankrupt:** This attribute tells whether the player has bankrupted or not. Bankrupted players cannot play in the game session anymore.

**private int atalarsRoomFreeCardNumber:** This attribute is the index of the card that enables a player to get out of Atalar’s Room. When players pick that card, they can use it whenever they want until the end of the game.

**private Color color:** This attribute is the color of the player. Each player has a unique color that is utilized for player UI purposes (border of player info, credit card of the player, next turn etc.).

**private boolean isInAtalarsRoom:** This attribute represents whether the player is in Atalar’s Room or not. If true, the player can use the atalarsRoomFreeCardNumber if they have that card. If not, the player can roll the dice and if both have equal values, they can leave Atalar’s Room.

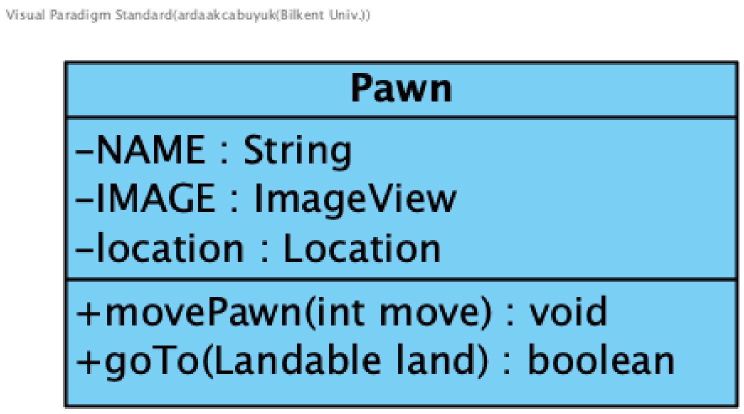
## Methods

**public boolean changeMoney(int amount):** This method is called whenever the player makes operations with money. The parameter amount is deduced or added to the balance of the player. The function returns true if successful.

**public boolean buyLandOrCafe(Buyable b):** This method is called whenever the player buys a Land or Cafe (Buyable). The Buyable is added to the ArrayList ownedLands. The function returns true if successful.

**public boolean sellLandOrCafe(Buyable b):** This method is called whenever the player sells a Land or Cafe (Buyable). The Buyable is removed from the ArrayList ownedLands. The function returns true if successful.

## Pawn Class


Visual Paradigm Standard(ardaakcabuyuk(Bilkent Univ.))
Pawn
-NAME : String
-IMAGE : ImageView
-location : Location
+movePawn(int move) : void
+goTo(Landable land) : boolean

### <u>Attributes</u>

**private final String NAME:** This attribute is the name of the Pawn.

**private final ImageView IMAGE:** This attribute is the image of the Pawn which will be displayed on the game board.

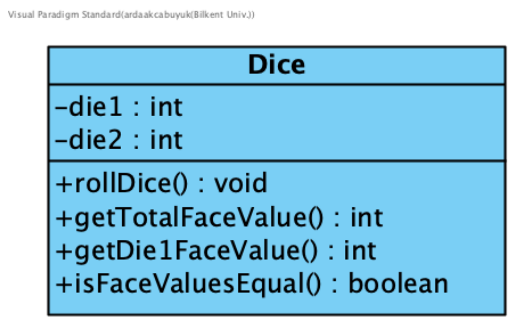
**private Location location:** This attribute is the location of the Pawn on the game board.

**Methods**

**public void movePawn(int move):** This method moves the pawn on the game board according to the move count that is passed as parameter.

**public boolean goTo(Landable land):** This method sends the Pawn to the Landable that is passed as the parameter. The function returns true if successful.

## Dice Class



Visual Paradigm Standard(ardaakcabuyuk(Bilkent Univ.))
Dice
-die1 : int
-die2 : int
+rollDice() : void
+getTotalFaceValue() : int
+getDie1FaceValue() : int
+isFaceValuesEqual() : boolean

**Attributes**

**private int die1:** This attribute is the value of the die1 (1-6).

**private int die2:** This attribute is the value of the die2 (1-6).

**Methods**

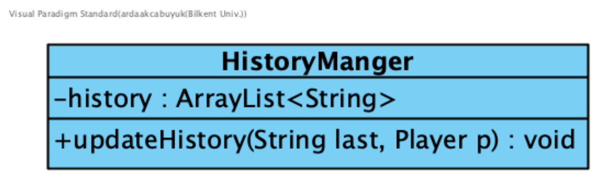
**public void rollDice():** This method rolls the dice. It basically generates a random number between 1 and 6 for each die.

**public int getTotalFaceValue():** This method sums up the values of die1 and die2 and returns it. It is used for moving the Pawn of the Player.

**public int getDie1FaceValue():** This method returns the value of die1, which will be used in single dice operations (e.g. choosing the player that starts the game).

**public boolean isFaceValuesEqual():** This method returns true if the values of die1 and die2 are the same. If true, either the player will roll the dice for one more turn, or if the player is in Atalar’s Room, they will leave it.

## HistoryManager Class


Visual Paradigm Standard(ardaakcabuyuk(Bilkent Univ.))
HistoryManger
-history : ArrayList<String>
+updateHistory(String last, Player p) : void

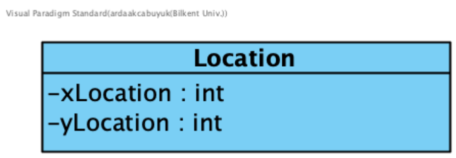
### Attributes

**private ArrayList<String> history:** This attribute holds the last three moves that have been done in the game, as sentences.

### Methods

**public void updateHistory(String last, Player p):** In each turn, this function is called and the most recent move is added to the history, while the oldest is removed from the history.

## Location Class

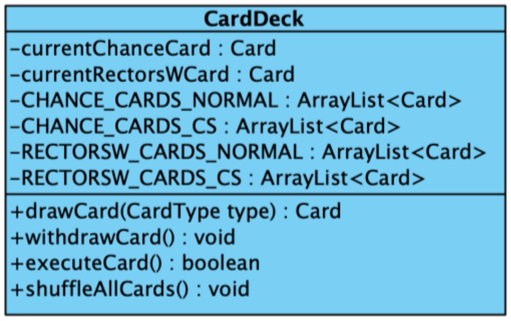


### Attributes

**private int xLocation:** This attribute holds the x location of whatever has a coordinate (e.g. Pawn, *Landable*).

**private int yLocation:** This attribute holds the x location of whatever has a coordinate (e.g. Pawn, *Landable*).

## CardDeck Class


CardDeck
-currentChanceCard : Card
-currentRectorsWCard : Card
-CHANCE_CARDS_NORMAL : ArrayList<Card>
-CHANCE_CARDS_CS : ArrayList<Card>
-RECTORSW_CARDS_NORMAL : ArrayList<Card>
-RECTORSW_CARDS_CS : ArrayList<Card>
+drawCard(CardType type) : Card
+withdrawCard() : void
+executeCard() : boolean
+shuffleAllCards() : void

### Attributes

**private Card currentChanceCard:** This attribute holds the Chance Card at the top of the deck which a player picks when landed on the according CardPlace.

**private Card currentRectorsWCard:** This attribute holds the Rector’s Whisper Card at the top of the deck which a player picks when landed on the according CardPlace.

**private final ArrayList<Card> CHANCE_CARDS_NORMAL:** This attribute holds the Chance Cards of the Buildings Mode.

**private final ArrayList<Card> CHANCE_CARDS_CS:** This attribute holds the Chance Cards of the CS Mode.

**private final ArrayList<Card> RECTORSW_CARDS_NORMAL:** This attribute holds the Rector’s Whisper Cards of the Buildings Mode.

**private final ArrayList<Card> RECTORSW_CARDS_CS:** This attribute holds the Rector’s Whisper Cards of the CS Mode.

## <u>Methods</u>

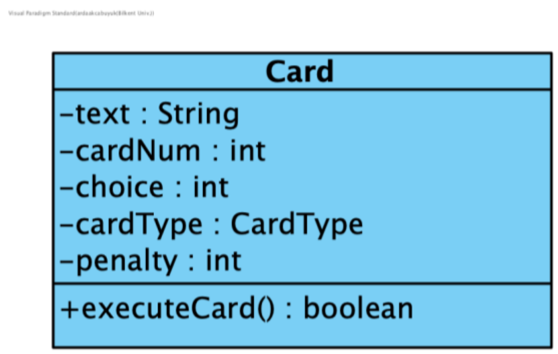
**public Card drawCard(CardType type):** This method returns the Card at the top (currentChanceCard or currentRectorsWCard, according to the type passed).

**public void withdrawCard():** This method puts the returned card end of the deck and makes current card null.

**public boolean executeCard():** This method executes the current card by calling executeCard() of that card and returns true. If no card is selected returns false.

**public void shuffleAllCards():** This method shuffles all the cards in the ArrayLists.

## Card Class



### Attributes

**private String text:** This attribute is the content (text) of the card.

**private int cardNum:** This attribute is the index of the card in the CardDeck.

**private int choice:** This attribute is the choice of the player in the choice-based cards.

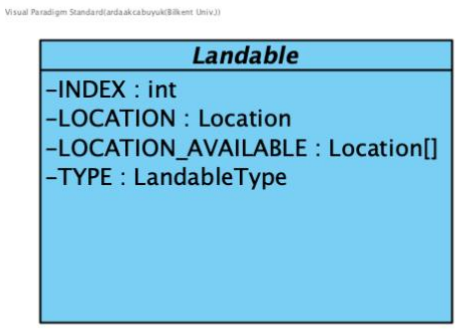
**private CardType cardType:** This attribute shows the type of the card. It is enum.

**private int penalty:** This attribute is the penalty or reward for the execution of the card (ex: It is your birthday, get 5.000 dollars).

### Methods

**public boolean executeCard():** This method executes the card and returns true. If it cannot be executed, returns false (ex: Player does not have sufficient money).

## Landable Class


Visual Paradigm Standard(ardaakcabuyuk(Bilkent Univ.))
Landable
-INDEX : int
-LOCATION : Location
-LOCATION_AVAILABLE : Location[]
-TYPE : LandableType

### Attributes

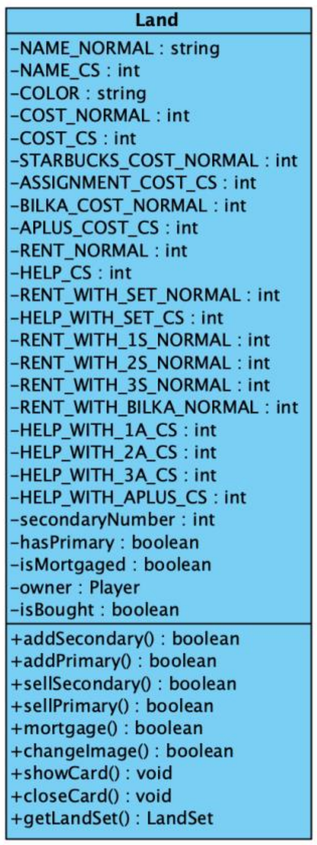
**private final int INDEX:** This attribute is the index of the Landable in the landableList ArrayList of GameManager.

**private final Location LOCATION:** This attribute is the location of the Landable on the game board.

**private final Location[] LOCATION_AVAILABLE:** This attribute is the Locations that Pawns could be placed on the Landable. If there are more than one Pawn on a Landable, this attribute helps to arrange the Pawns.

**private LandableType TYPE:** This attribute holds the type of the Landable (e.g. Cafe, Land, CardPlace etc.)

## Land Class



### <u>Attributes</u>

**private final String NAME_NORMAL:** This attribute is the name of the Land (Building) in Normal Mode.

**private final int NAME_CS:** This attribute is the name of the Land (Course) in CS Mode.

**private final String COLOR:** This attribute is the color of the Land which represents the ColorSet that the Land belongs.

**private final int COST_NORMAL:** This attribute is the cost of the Land (Building) for Normal Mode.

**private final int COST_CS:** This attribute is the cost of the Land (Course) for CS Mode.

**private final int STARBUCKS_COST_NORMAL:** This attribute is the cost of building Starbucks on the Land (Building) for Normal Mode.

**private final int ASSIGNMENT_COST_CS:** This attribute is the cost of doing Assignment for the Land (Course) for CS Mode.

**private final int BILKA_COST_NORMAL:** This attribute is the cost of building Bilka on the Land (Building) for Normal Mode.

**private final int APLUS_COST_CS:** This attribute is the cost of getting A+ for the Land (Course) for CS Mode.

**private final int RENT_NORMAL:** This attribute is the rent of the Land (Building) with no Starbucks or Bilkas for Normal Mode.

**private final int HELP_CS:** This attribute is the help cost of the Land (Course) with no Assignments or A+s for CS Mode.

**private final int RENT_WITH_SET_NORMAL:** This attribute is the rent of the Land (Building) with all ColorSet bought for Normal Mode.

**private final int HELP_WITH_SET_CS:** This attribute is the help cost of the Land (Course) with all ColorSet bought for CS Mode.

**private final int RENT_WITH_1S_NORMAL:** This attribute is the rent of the Land (Building) with 1 Starbucks for Normal Mode.

**private final int RENT_WITH_2S_NORMAL:** This attribute is the rent of the Land (Building) with 2 Starbuckses for Normal Mode.

**private final int RENT_WITH_3S_NORMAL:** This attribute is the rent of the Land (Building) with 3 Starbuckses for Normal Mode.

**private final int RENT_WITH_BILKA_NORMAL:** This attribute is the rent of the Land (Building) with Bilka for Normal Mode.

**private final int HELP_WITH_1A_CS:** This attribute is the help cost of the Land (Course) with 1 Assignment for CS Mode.

**private final int HELP_WITH_2A_CS:** This attribute is the help cost of the Land (Course) with 2 Assignments for CS Mode.

**private final int HELP_WITH_3A_CS:** This attribute is the help cost of the Land (Course) with 3 Assignments for CS Mode.

**private final int HELP_WITH_APLUS_CS:** This attribute is the help cost of the Land (Course) with A+ for CS Mode.

**private int secondaryNumber:** This attribute is the number of starbucks on the Land (number of assignments in CS Mode).

**private boolean hasPrimary:** This attribute tells whether the Land has Bilka (or A+ in CS Mode) on it or not.

**private boolean isMortgaged:** This attribute tells whether the Land is mortgaged or not (whether the course is withdrawn or not in CS Mode).

**private Player owner:** This attribute is the Player who owns the Land.

**private boolean isBought:** This attribute tells whether the Land is bought or not.

## Methods

**public boolean addSecondary():** This method adds a Starbucks on the Land for Normal Mode or adds an Assignment for CS Mode. Return value is a success indicating boolean.

**public boolean addPrimary():** This method adds a Bilka on the Land for Normal Mode or adds an A+ for CS Mode. Return value is a success indicating boolean.

**public boolean sellSecondary():** This method removes a Starbucks on the Land for Normal Mode or removes an Assignment for CS Mode. Return value is a success indicating boolean.

**public boolean sellPrimary():** This method removes a Bilka on the Land for Normal Mode or removes an A+ for CS Mode. Return value is a success indicating boolean.

**public boolean mortgage():** This method mortgages the land. Return value is a success indicating boolean.

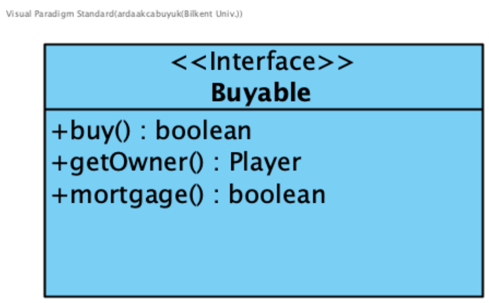
**public boolean changeImage():** This method changes the image of the land. It will help us to initialize the board.

**public void showCard():** This method shows the card of the Land when a player lands on.

**public void closeCard():** This method closes the card of the Land.

**public LandSet getLandSet():** This method returns the LandSet which the Land belongs to. It will help us to point to its land set (color set).

## Buyable Interface



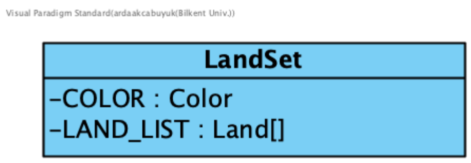
### <u>Methods</u>

**public boolean buy():** This method buys the Buyable for the player.

**public Player getOwner():** This method returns the owner of the Buyable.

**public boolean mortgage():** This method mortgages the Buyable.

## LandSet Class

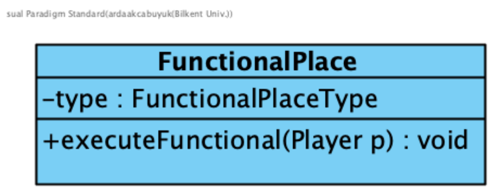


### <u>Attributes</u>

**private final Color COLOR:** This attribute is the color of the LandSet.

**private final Land[] LAND_LIST:** This attribute holds the lands that this LandSet has.

## FunctionalPlace Class


sual Paradigm Standard(ardaakcabuyuk(Bilkent Univ.))
FunctionalPlace
–type : FunctionalPlaceType
+executeFunctional(Player p) : void

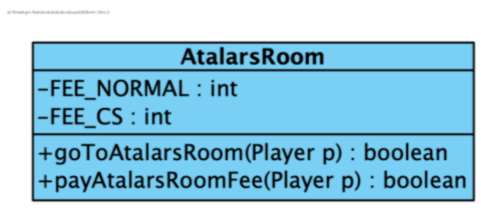
### Attributes

**private FunctionalPlaceType type:** This attribute indicates the type of the FunctionalPlace (e.g. Nizamiye, Free Parking, Go to Atalar’s Room).

### Methods

**public void executeFunctional():** This method executes the function of the FunctionalPlace and updates player attributes (e.g. player earns money when passed Nizamiye).

## AtalarsRoom Class


AtalarsRoom
–FEE_NORMAL : int
–FEE_CS : int
+goToAtalarsRoom(Player p) : boolean
+payAtalarsRoomFee(Player p) : boolean

### Attributes

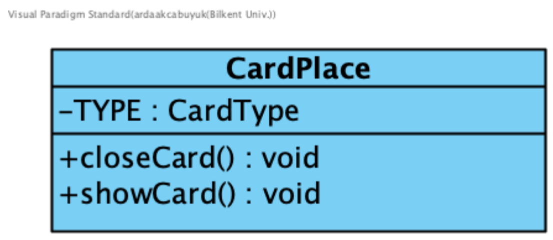
**private int fee:** This attribute is the fee to pay to get out from Atalar’s Room.

## Methods

**public boolean goToAtalarsRoom(Player p):** This method sends the Player passed to Atalar’s Room.

**public boolean payAtalarsRoomFee(Player p):** This method deduces the fee from the Player that is about to leave Atalar’s Room.

## CardPlace Class



## Attributes

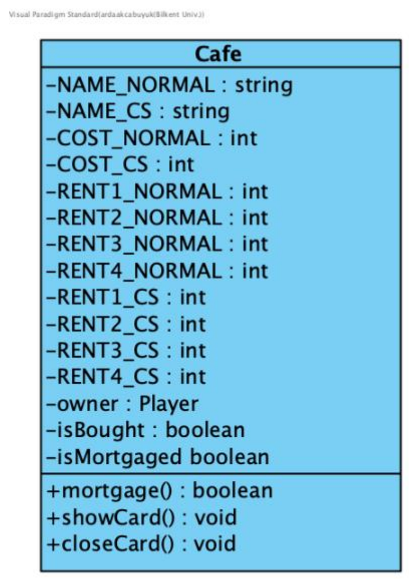
**private final CardType TYPE:** This attribute is the type of the CardPlace (Chance/Rector’s Whisper)

## Methods

**public void showCard():** This method displays the popup of the picked Card on the screen.

**public void closeCard():** This method closes the picked Card.

## Cafe Class



### <u>Attributes</u>

**private final String NAME_NORMAL:** This attribute is the name of the Cafe on Normal Mode.

**private final int NAME_CS:** This attribute is the name of the Cafe on CS Mode.

**private final int COST_NORMAL:** This attribute is the cost of the Cafe on Normal Mode.

**private final int COST_CS:** This attribute is the cost of the Cafe on CS Mode.

**private final int RENT1_NORMAL:** This attribute is the rent of the Cafe on Normal Mode, if the owner has 1 cafe.

**private final int RENT2_NORMAL:** This attribute is the rent of the Cafe on Normal Mode, if the owner has 2 cafes.

**private final int RENT3_NORMAL:** This attribute is the rent of the Cafe on Normal Mode, if the owner has 3 cafes.

**private final int RENT4_NORMAL:** This attribute is the rent of the Cafe on Normal Mode, if the owner has all of the cafes.

**private final int HELP1_CS:** This attribute is the help cost of the Cafe on CS Mode, if the owner has 1 cafe.

**private final int HELP2_CS:** This attribute is the help cost of the Cafe on CS Mode, if the owner has 2 cafes.

**private final int HELP3_CS:** This attribute is the help cost of the Cafe on CS Mode, if the owner has 3 cafes.

**private final int HELP4_CS:** This attribute is the help cost of the Cafe on CS Mode, if the owner has all of the cafes.

**private Player owner:** This attribute is the Player that owns this Cafe.

**private boolean isBought:** This attribute tells whether the Cafe is bought or not.

**private boolean isMortgaged:** This attribute tells whether the Cafe is mortgaged or not.

## Methods

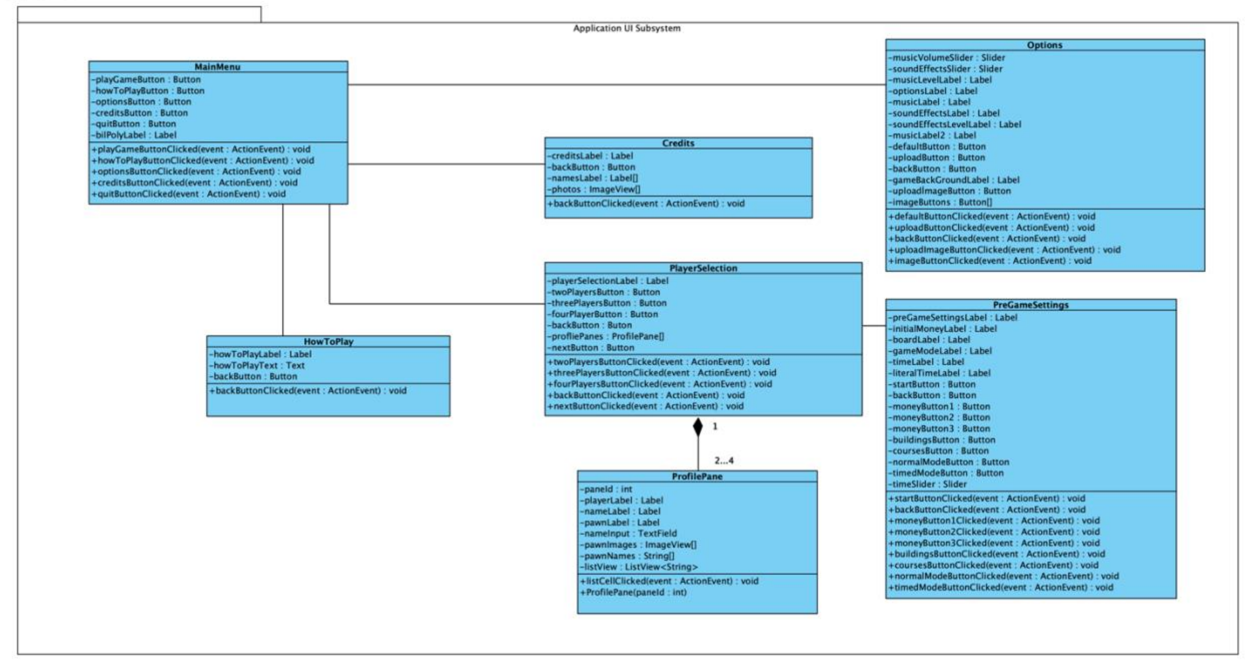
**public boolean mortgage():** This method mortgages the Cafe. Returns true if the method succeeds.

**public void showCard():** This method shows the popup of the Cafe when a Player lands on it.

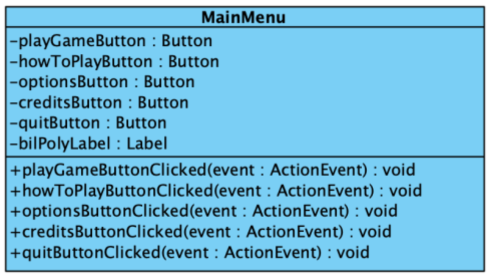
**public void closeCard():** This attribute closes the popup of the Cafe.

## 3.3 User Interface Subsystem

### 3.3.1 Application UI Subsystem


Application UI Subsystem
MainMenu
-playGameButton : Button
-howToPlayButton : Button
-optionsButton : Button
-creditsButton : Button
-quitButton : Button
-bilPolyLabel : Label
+playGameButtonClicked(event : ActionEvent) : void
+howToPlayButtonClicked(event : ActionEvent) : void
+optionsButtonClicked(event : ActionEvent) : void
+creditsButtonClicked(event : ActionEvent) : void
+quitButtonClicked(event : ActionEvent) : void
Credits
-creditsLabel : Label
-backButton : Button
-namesLabel : Label[]
-photos : ImageView[]
+backButtonClicked(event : ActionEvent) : void
Options
-musicVolumeSlider : Slider
-soundEffectsSlider : Slider
-musicLevelLabel : Label
-optionsLabel : Label
-musicLabel : Label
-soundEffectsLabel : Label
-soundEffectsLevelLabel : Label
-musicLabel2 : Label
-defaultButton : Button
-uploadButton : Button
-backButton : Button
-gameBackGroundLabel : Label
-uploadImageButton : Button
-imageButtons : Button[]
+defaultButtonClicked(event : ActionEvent) : void
+uploadButtonClicked(event : ActionEvent) : void
+backButtonClicked(event : ActionEvent) : void
+uploadImageButtonClicked(event : ActionEvent) : void
+imageButtonClicked(event : ActionEvent) : void
HowToPlay
-howToPlayLabel : Label
-howToPlayText : Text
-backButton : Button
+backButtonClicked(event : ActionEvent) : void
PlayerSelection
-playerSelectionLabel : Label
-twoPlayersButton : Button
-threePlayersButton : Button
-fourPlayerButton : Button
-backButton : Buton
-profliePanes : ProfilePane[]
-nextButton : Button
+twoPlayersButtonClicked(event : ActionEvent) : void
+threePlayersButtonClicked(event : ActionEvent) : void
+fourPlayersButtonClicked(event : ActionEvent) : void
+backButtonClicked(event : ActionEvent) : void
+nextButtonClicked(event : ActionEvent) : void
1
2...4
ProfilePane
-paneId : int
-playerLabel : Label
-nameLabel : Label
-pawnLabel : Label
-nameInput : TextField
-pawnImages : ImageView[]
-pawnNames : String[]
-listView : ListView<String>
+listCellClicked(event : ActionEvent) : void
+ProfilePane(paneId : int)
PreGameSettings
-preGameSettingsLabel : Label
-initialMoneyLabel : Label
-boardLabel : Label
-gameModeLabel : Label
-timeLabel : Label
-literalTimeLabel : Label
-startButton : Button
-backButton : Button
-moneyButton1 : Button
-moneyButton2 : Button
-moneyButton3 : Button
-buildingsButton : Button
-coursesButton : Button
-normalModeButton : Button
-timedModeButton : Button
-timeSlider : Slider
+startButtonClicked(event : ActionEvent) : void
+backButtonClicked(event : ActionEvent) : void
+moneyButton1Clicked(event : ActionEvent) : void
+moneyButton2Clicked(event : ActionEvent) : void
+moneyButton3Clicked(event : ActionEvent) : void
+buildingsButtonClicked(event : ActionEvent) : void
+coursesButtonClicked(event : ActionEvent) : void
+normalModeButtonClicked(event : ActionEvent) : void
+timedModeButtonClicked(event : ActionEvent) : void

### MainMenu Class


MainMenu
-playGameButton : Button
-howToPlayButton : Button
-optionsButton : Button
-creditsButton : Button
-quitButton : Button
-bilPolyLabel : Label
+playGameButtonClicked(event : ActionEvent) : void
+howToPlayButtonClicked(event : ActionEvent) : void
+optionsButtonClicked(event : ActionEvent) : void
+creditsButtonClicked(event : ActionEvent) : void
+quitButtonClicked(event : ActionEvent) : void

#### <u>Attributes</u>

**private Button playGameButton:** This is a button instance for play game button.

**private Button howToPlayButton:** This is a button instance for how to play button.

**private Button optionsButton:** This is a button instance for options button.

**private Button creditsButton:** This is a button instance for credits button.

**private Button quitButton:** This is a button instance for quit game button.

**private Label bilPolyLabel:** This is a label instance for the label BilPoly.

## Methods

**public playGameButtonClicked(event : ActionEvent):** This is a listener method for the play game button. It is used to navigate to the player selection.

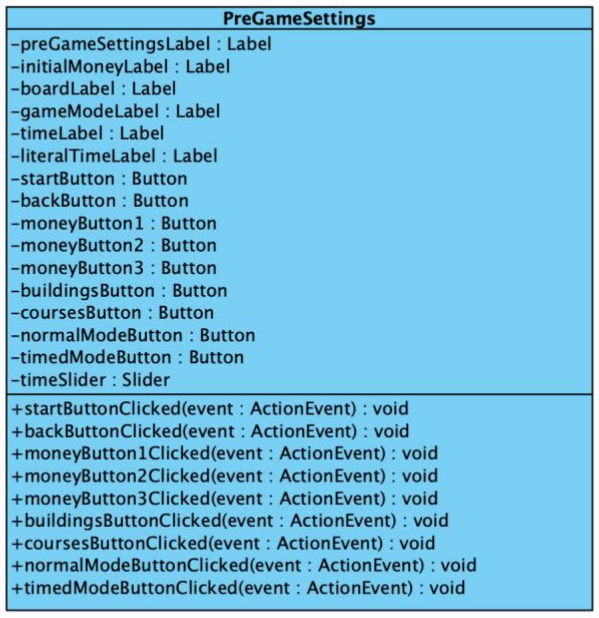
**public howToPlayButtonClicked(event : ActionEvent):** This is a listener method for the how to play button. It is used to navigate to how to play.

**public optionsButtonClicked(event : ActionEvent):**  This is a listener method for the options button. It is used to navigate to options.

**public creditsButtonClicked(event : ActionEvent):**This is a listener method for the credits button. It is used to navigate to credits.

**public quitButtonClicked(event : ActionEvent):**This is a listener method for the credits button. It is used to quit the game.

## PreGameSettings Class


PreGameSettings
-preGameSettingsLabel : Label
-initialMoneyLabel : Label
-boardLabel : Label
-gameModeLabel : Label
-timeLabel : Label
-literalTimeLabel : Label
-startButton : Button
-backButton : Button
-moneyButton1 : Button
-moneyButton2 : Button
-moneyButton3 : Button
-buildingsButton : Button
-coursesButton : Button
-normalModeButton : Button
-timedModeButton : Button
-timeSlider : Slider
+startButtonClicked(event : ActionEvent) : void
+backButtonClicked(event : ActionEvent) : void
+moneyButton1Clicked(event : ActionEvent) : void
+moneyButton2Clicked(event : ActionEvent) : void
+moneyButton3Clicked(event : ActionEvent) : void
+buildingsButtonClicked(event : ActionEvent) : void
+coursesButtonClicked(event : ActionEvent) : void
+normalModeButtonClicked(event : ActionEvent) : void
+timedModeButtonClicked(event : ActionEvent) : void

### Attributes

**private Label preGameSettingsLabel :** This is a Label instance for the Pre-Game Settings label.

**private Label initialMoneyLabel :** This is a Label instance for the Initial Money label.

**private Label boardLabel :** This is a Label instance for the board label.

**private Label gameModeLabel :** This is a Label instance for the game mode label.

**private Label timeLabel :** This is a Label instance for the time mode label.

**private Label literalTimeLabel :** This is a label instance to show the time limit that is decided by the user.

**private Button startButton :** This is a button instance for start button. After clicking this button, the game will start.

**private Button backButton :** This is a button instance for back button. It is used to go back to the Player Selection.

**private Button moneyButton1 :** This is a button instance to choose the money amount that everyone will have at the start of the game.

**private Button moneyButton2 :** This is a button instance to choose the money amount that everyone will have at the start of the game.

**private Button moneyButton3 :** This is a button instance to choose the money amount that everyone will have at the start of the game.

**private Button buildingsButton:** This is a button instance to let the user decide if the Game Board Mode is in the Bilkent Buildings Mode when it is clicked.

**private Button coursesButton:T**his is a button instance to let the user decide if the Game Board Mode is in the Bilkent CS Mode when it is clicked.

**private Button normalModeButton:**This is a button instance to let the user decide if the Time Mode is in the Normal Mode when it is clicked.

**private Button timedModeButton:**This is a button instance to let the user decide if the Time Mode is in the Timed Mode when it is clicked.

**private Slider timeSlider:** This is a slider instance to let the user decide the time limit.

**<u>Methods</u>**

**public void startButtonClicked(event : ActionEvent):** This is a listener method for start button.

**public void backButtonClicked(event : ActionEvent):**This is a listener method for back button.

**public void moneyButton1Clicked(event : ActionEvent):**This is a listener method for the money button at the top.

**public void moneyButton2Clicked(event : ActionEvent):**This is a listener method for the money button at the middle.

**public void moneyButton3Clicked(event : ActionEvent):**This is a listener method for the money button at the bottom.

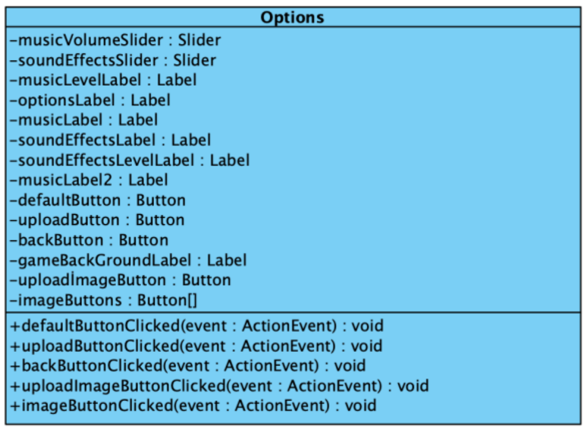
**public void buildingsButtonClicked(event : ActionEvent):**This is a listener method for buildings button.

**public void coursesButtonClicked(event : ActionEvent):**This is a listener method for courses button.

**public void normalModeButtonClicked(event : ActionEvent):**This is a listener method for normal mode button.

**public void timedModeButtonClicked(event : ActionEvent):**This is a listener method for timed mode button.

## Options Class


Options
-musicVolumeSlider : Slider
-soundEffectsSlider : Slider
-musicLevelLabel : Label
-optionsLabel : Label
-musicLabel : Label
-soundEffectsLabel : Label
-soundEffectsLevelLabel : Label
-musicLabel2 : Label
-defaultButton : Button
-uploadButton : Button
-backButton : Button
-gameBackGroundLabel : Label
-uploadImageButton : Button
-imageButtons : Button[]
+defaultButtonClicked(event : ActionEvent) : void
+uploadButtonClicked(event : ActionEvent) : void
+backButtonClicked(event : ActionEvent) : void
+uploadImageButtonClicked(event : ActionEvent) : void
+imageButtonClicked(event : ActionEvent) : void

### Attributes

**private Slider musicVolumeSlider:** This is a Slider instance for the music volume.

**private Slider soundEffectsSlider:** This is a Slider instance for the sound effects volume.

**private Label musicLevelLabel:** This is a Label instance for the music level label.

**private Label optionsLabel:** This is a Label instance for the options label.

**private Label musicLabel:** This is a Label instance for the music level label.

**private Label soundEffectsLabel:** This is a Label instance for the sound effects label.

**private Label soundEffectsLevelLabel:** This is a Label instance for the sound effects level label.

**private Label musicLabel2:** This is a Label instance for the second music label.

**private Button defaultButton:** This is a Button instance for the default button. It is used to select the default music.

**private Button uploadButton:** This is a Button instance for the upload button. It is used to upload music.

**private Button backButton:** This is a Button instance for the back button. It is used to navigate the main menu.

**private Label gameBackGroundLabel:** This is a label instance for the label “Game Background”.

**private Button uploadImageButton:** This is a Button instance for the upload image button. It is used to upload tracks from the local files.

**private Button imageButtons[]:** This is an instance of an array of Button. It holds the default images’ buttons.

## Methods

**public void defaultButtonClicked(event: ActionEvent):** This is a listener method for the default button.

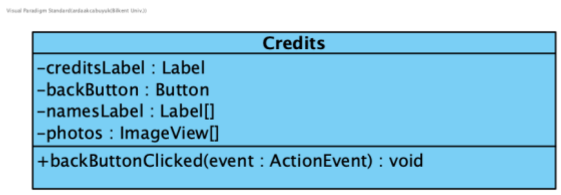
**public void uploadButtonClicked(event: ActionEvent):** This is a listener method for the upload button.

**public void backButtonClicked(event: ActionEvent):** This is a listener method for the back button.

**public void uploadImageButtonClicked(event: ActionEvent):** This is a listener method for the upload image mode button.

**public void imageButtonClicked(event: ActionEvent):** This is a listener method for the image button.

## Credits Class



### Attributes

**private Label creditsLabel:** This is a Label instance for the credits label.

**private Button backButton:** This is a Button instance for the back button.

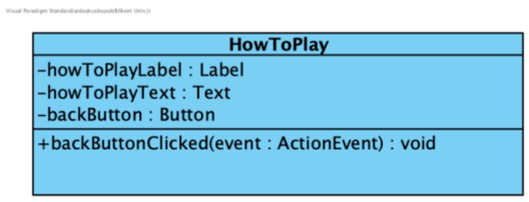
**private Label namesLabel[]:** This is an instance of an array of Label for the names of the developers.

**private ImageView photos[]:** This is an instance of an array of ImageView for the photos of the developers.

### Methods

**public void backButtonClicked(event: ActionEvent):** This is a listener method for the back button. It is used to navigate to the main menu.

## HowToPlay Class



### Attributes

**private Label howToPlayLabel:** This is a Label instance for the how to play label.

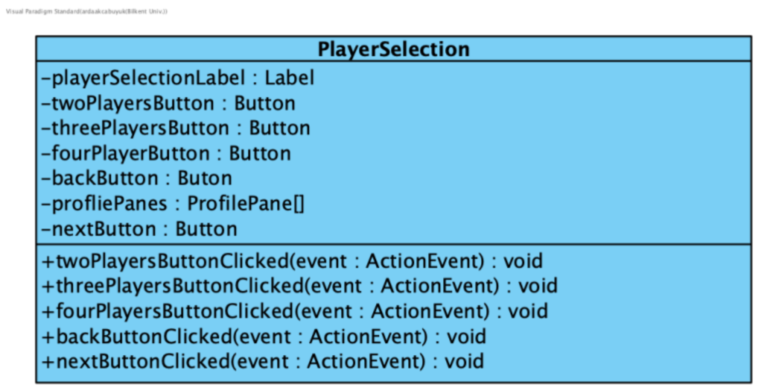
**private Text howToPlayText:** This is a Text instance that holds the how to play text.

**private Button backButton:** This is a Button instance for the back button.

### Methods

**public void backButtonClicked(event: ActionEvent):** This is a listener method for the back button. It is used to navigate to the main menu.

## PlayerSelection Class


PlayerSelection
-playerSelectionLabel : Label
-twoPlayersButton : Button
-threePlayersButton : Button
-fourPlayerButton : Button
-backButton : Buton
-profliePanes : ProfilePane[]
-nextButton : Button
+twoPlayersButtonClicked(event : ActionEvent) : void
+threePlayersButtonClicked(event : ActionEvent) : void
+fourPlayersButtonClicked(event : ActionEvent) : void
+backButtonClicked(event : ActionEvent) : void
+nextButtonClicked(event : ActionEvent) : void

### Attributes

**private Label playerSelectionLabel:** This is a Label instance for the player selection label.

**private Button twoPlayersButton:** This is a Button instance for the two players button. It is used when there are two players.

**private Button threePlayersButton:** This is a Button instance for the three players button. It is used when there are three players.

**private Button fourPlayersButton:** This is a Button instance for the four players button. It is used when there are four players.

**private Button backButton:** This is a Button instance for the back button.

**private Button nextButton:** This is a Button instance for the next button.

**private ProfilePane profilePanes[]:** This is a ProfilePane array.

### Methods

**public void twoPlayersButtonClicked(event: ActionEvent):** This is a listener method for the two players button.

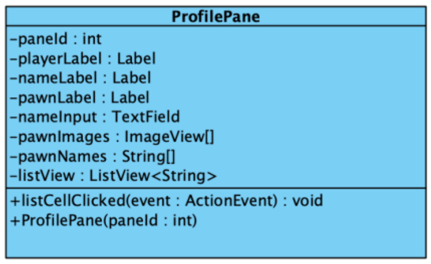
**public void threePlayersButtonClicked(event: ActionEvent):** This is a listener method for the three players button.

**public void fourPlayersButtonClicked(event: ActionEvent):** This is a listener method for the four players button.

**public void backButtonClicked(event: ActionEvent):** This is a listener method for the back button. It is used to navigate to the main menu.

**public void nextButtonClicked(event: ActionEvent):** This is a listener method for the next button. It is used to navigate to the pregame settings.

## ProfilePane Class


ProfilePane
-paneId : int
-playerLabel : Label
-nameLabel : Label
-pawnLabel : Label
-nameInput : TextField
-pawnImages : ImageView[]
-pawnNames : String[]
-listView : ListView<String>
+listCellClicked(event : ActionEvent) : void
+ProfilePane(paneId : int)

### Attributes

**private int paneId:** An integer variable that keeps the pane's id. Pane id can take values 1-4.

**private Label playerLabel:** This is a Label instance for the player label.

**private Label nameLabel:**This is a Label instance for the name label.

**private Label pawnLabel:** This is a Label instance for the pawn label.

**private TextField nameInput:** This is a TextField instance for the player’s name input.

**private ImageView pawnImages[]:** This is an instance of an array of ImageView for the images of the pawns.

**private String pawnNames[]:** This is a String array that holds pawn names.

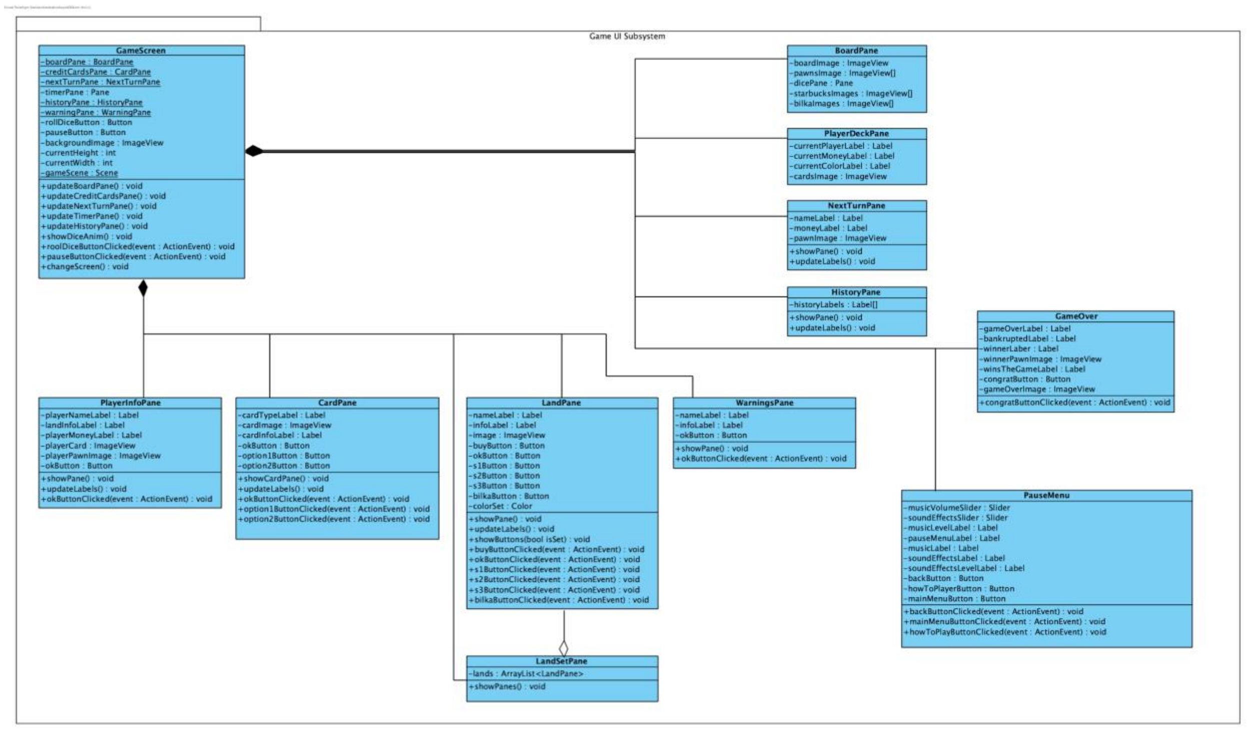
**private ListView<String> listView:** This is a ListView instance, it is used to create the pawn selection unit.

## Methods

**public void listCellClicked(event: ActionEvent):** This is the listener method to know if an item is clicked in the listView.

**public ProfilePane(paneId: int):** This is the constructor of ProfilePane class, takes panelId as the parameter when initialized .

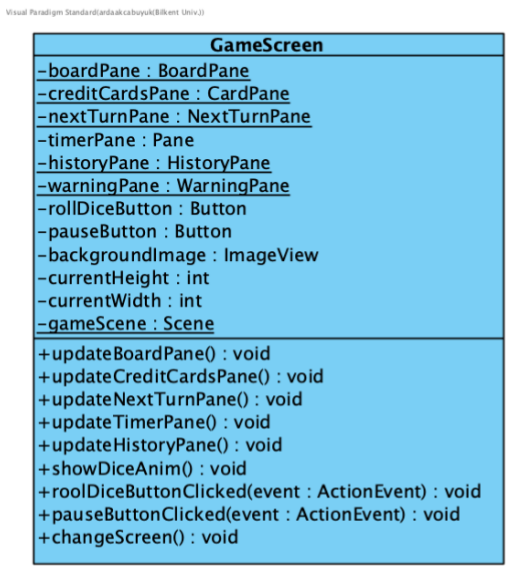
## 3.3.2 Game UI Subsystem


Game UI Subsystem
GameScreen
-boardPane : BoardPane
-creditCardsPane : CardPane
-nextTurnPane : NextTurnPane
-timerPane : Pane
-historyPane : HistoryPane
-warningPane : WarningPane
-rollDiceButton : Button
-pauseButton : Button
-backgroundImage : ImageView
-currentHeight : int
-currentWidth : int
-gameScene : Scene
+updateBoardPane() : void
+updateCreditCardsPane() : void
+updateNextTurnPane() : void
+updateTimerPane() : void
+updateHistoryPane() : void
+showDiceAnim() : void
+roolDiceButtonClicked(event : ActionEvent) : void
+pauseButtonClicked(event : ActionEvent) : void
+changeScreen() : void
BoardPane
-boardImage : ImageView
-pawnsImage : ImageView[]
-dicePane : Pane
-starbucksImages : ImageView[]
-bilkaImages : ImageView[]
PlayerDeckPane
-currentPlayerLabel : Label
-currentMoneyLabel : Label
-currentColorLabel : Label
-cardsImage : ImageView
NextTurnPane
-nameLabel : Label
-moneyLabel : Label
-pawnImage : ImageView
+showPane() : void
+updateLabels() : void
HistoryPane
-historyLabels : Label[]
+showPane() : void
+updateLabels() : void
GameOver
-gameOverLabel : Label
-bankruptedLabel : Label
-winnerLaber : Label
-winnerPawnImage : ImageView
-winsTheGameLabel : Label
-congratButton : Button
-gameOverImage : ImageView
+congratButtonClicked(event : ActionEvent) : void
PlayerInfoPane
-playerNameLabel : Label
-landInfoLabel : Label
-playerMoneyLabel : Label
-playerCard : ImageView
-playerPawnImage : ImageView
-okButton : Button
+showPane() : void
+updateLabels() : void
+okButtonClicked(event : ActionEvent) : void
CardPane
-cardTypeLabel : Label
-cardImage : ImageView
-cardInfoLabel : Label
-okButton : Button
-option1Button : Button
-option2Button : Button
+showCardPane() : void
+updateLabels() : void
+okButtonClicked(event : ActionEvent) : void
+option1ButtonClicked(event : ActionEvent) : void
+option2ButtonClicked(event : ActionEvent) : void
LandPane
-nameLabel : Label
-infoLabel : Label
-image : ImageView
-buyButton : Button
-okButton : Button
-s1Button : Button
-s2Button : Button
-s3Button : Button
-bilkaButton : Button
-colorSet : Color
+showPane() : void
+updateLabels() : void
+showButtons(bool isSet) : void
+buyButtonClicked(event : ActionEvent) : void
+okButtonClicked(event : ActionEvent) : void
+s1ButtonClicked(event : ActionEvent) : void
+s2ButtonClicked(event : ActionEvent) : void
+s3ButtonClicked(event : ActionEvent) : void
+bilkaButtonClicked(event : ActionEvent) : void
WarningsPane
-nameLabel : Label
-infoLabel : Label
-okButton : Button
+showPane() : void
+okButtonClicked(event : ActionEvent) : void
PauseMenu
-musicVolumeSlider : Slider
-soundEffectsSlider : Slider
-musicLevelLabel : Label
-pauseMenuLabel : Label
-musicLabel : Label
-soundEffectsLabel : Label
-soundEffectsLevelLabel : Label
-backButton : Button
-howToPlayerButton : Button
-mainMenuButton : Button
+backButtonClicked(event : ActionEvent) : void
+mainMenuButtonClicked(event : ActionEvent) : void
+howToPlayButtonClicked(event : ActionEvent) : void
LandSetPane
-lands : ArrayList<LandPane>
+showPanes() : void

Game UI Subsystem consists of 12 classes that assemble the in-game user interface.

The main class of this subsystem is GameScreen, which contains Game Over and Pause screens and Panes that compose the in-game screen. The Panes are PlayerInfoPane which shows the information about a Player, CardPane which shows the content of the picked Card, LandPane which pops on the screen when a Player lands on a Buyable, LandSetPane which shows all Lands in the same LandSet, WarningsPane for warning popups, BoardPane which contains the game board, PlayerDeckPane which displays the credit cards of the Players, NextTurnPane which shows the Player who has the next turn, and HistoryPane which shows the history.

## GameScreen Class


Visual Paradigm Standard(ardaakcabuyuk(Bilkent Univ.))
GameScreen
-boardPane : BoardPane
-creditCardsPane : CardPane
-nextTurnPane : NextTurnPane
-timerPane : Pane
-historyPane : HistoryPane
-warningPane : WarningPane
-rollDiceButton : Button
-pauseButton : Button
-backgroundImage : ImageView
-currentHeight : int
-currentWidth : int
-gameScene : Scene
+updateBoardPane() : void
+updateCreditCardsPane() : void
+updateNextTurnPane() : void
+updateTimerPane() : void
+updateHistoryPane() : void
+showDiceAnim() : void
+roolDiceButtonClicked(event : ActionEvent) : void
+pauseButtonClicked(event : ActionEvent) : void
+changeScreen() : void

### Attributes

**private BoardPane boardPane:** This attribute is the Pane in which the game board will be placed.

**private NextTurnPane nextTurnPane:** This attribute is the Pane in which the next turn panel will be placed.

**private Pane timerPane:** This attribute is the Pane in which the Timer will count down in Timed Mode.

**private HistoryPane historyPane:** This attribute is the Pane in which the history information will be held.

**private WarningPane warningPane:** This attribute is the Pane in which the warning information will be placed. Warning popups will contain this Pane.

**private Button rollDiceButton:** This attribute is the Button which rolls the dice for the player who has the turn.

**private Button pauseButton:** This attribute is the pause button at the top-right of the screen.

**private ImageView backgroundImage:** This attribute is the background image of the game screen (which was the B Building on the mockups).

**private int currentHeight:** This attribute is the height of the game screen which changes from computer to computer.

**private int currentWidth:** This attribute is the width of the game screen which changes from computer to computer.

**private Scene gameScene:** This attribute is the game scene that all the Panes will be placed in.

## <u>Methods</u>

**public void updateBoardPane():** This method updates the BoardPane, therefore displays the changes on the Pane.

**public void updateCreditCardsPane():** This method updates the CreditCardsPane after each turn.

**public void updateNextTurnPane():** This method updates the NextTurnPane after each turn.

**public void updateTimerPane():** This method updates the TimerPane each second.

**public void updateHistoryPane():** This method updates the HistoryPane after each turn.

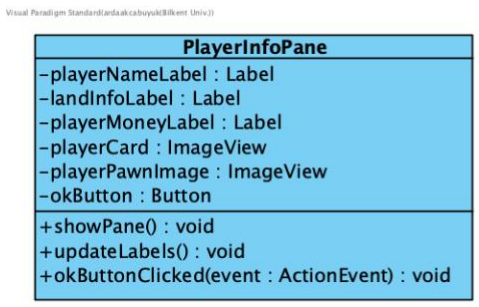
**public void showDiceAnim():** This method shows the dice animation in the middle of the game board when rolled.

**public void rollDiceButtonClicked(event : ActionEvent):** This listener method is called when the rollDiceButton is clicked.

**public void pauseButtonClicked(event : ActionEvent):** This listener method is called when the pauseButton is clicked.

**public void changeScreen():** This method calls the ScreenManager class to change the screen, if necessary.

## PlayerInfoPane Class


Visual Paradigm Standard(ardaakcabuyuk(Bilkent Univ.))
PlayerInfoPane
-playerNameLabel : Label
-landInfoLabel : Label
-playerMoneyLabel : Label
-playerCard : ImageView
-playerPawnImage : ImageView
-okButton : Button
+showPane() : void
+updateLabels() : void
+okButtonClicked(event : ActionEvent) : void

### <u>Attributes</u>

**private Label playerNameLabel:** This attribute is the name label of a player.

**private Label landInfoLabel:** This attribute is the land list of a player.

**private Label playerMoneyLabel:** This attribute is the balance of a player.

**private ImageView playerCard:** This attribute is the credit card image of a player.

**private ImageView playerPawnImage:** This attribute is the pawn image of a player.

**private Button okButton:** This attribute is the ok button that closes the pane.

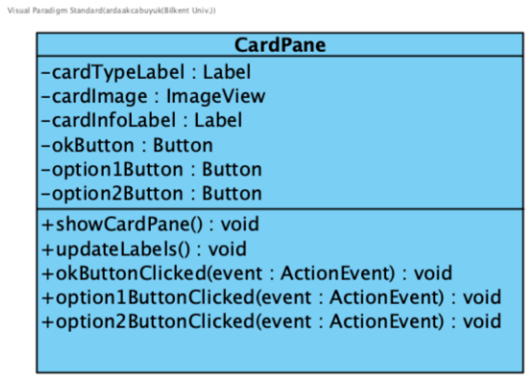
### Methods

**public void showPane():** This method shows the player info pane.

**public void updateLabels():** This method updates the labels in the pane.

**public void okButtonClicked(ActionEvent event):** This method is the listener of the ok button.

## CardPane Class


Visual Paradigm Standard(ardaakcabuyuk(Bilkent Univ.))
CardPane
-cardTypeLabel : Label
-cardImage : ImageView
-cardInfoLabel : Label
-okButton : Button
-option1Button : Button
-option2Button : Button
+showCardPane() : void
+updateLabels() : void
+okButtonClicked(event : ActionEvent) : void
+option1ButtonClicked(event : ActionEvent) : void
+option2ButtonClicked(event : ActionEvent) : void

### Attributes

**private Label cardTypeLabel:** This attribute is the card type.

**private Label cardInfoLabel:** This attribute is a text about the contents of the card.

**private ImageView cardImage:** This attribute shows a picture about the contents of the card.

**private Button okButton:** This attribute is the ok button that closes the pane.

**private Button option1Button:** This attribute  is a button created so that the player can choose one of the choice.

**private Button option2Button:** This attribute  is a button created so that the player can choose one of the choice.

## Methods

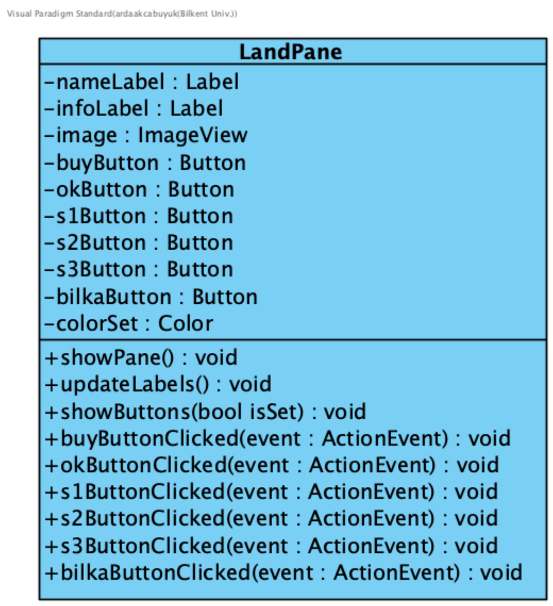
**public void showCardPane():** This method shows the card pane.

**public void updateLabels():** This method updates the labels in the pane.

**public void option1ButtonClicked(ActionEvent event):** This method is the listener of the option1 button.

**public void option2ButtonClicked(ActionEvent event):** This method is the listener of the option2 button.

## LandPane Class


Visual Paradigm Standard(ardaakcabuyuk(Bilkent Univ.))
LandPane
-nameLabel : Label
-infoLabel : Label
-image : ImageView
-buyButton : Button
-okButton : Button
-s1Button : Button
-s2Button : Button
-s3Button : Button
-bilkaButton : Button
-colorSet : Color
+showPane() : void
+updateLabels() : void
+showButtons(bool isSet) : void
+buyButtonClicked(event : ActionEvent) : void
+okButtonClicked(event : ActionEvent) : void
+s1ButtonClicked(event : ActionEvent) : void
+s2ButtonClicked(event : ActionEvent) : void
+s3ButtonClicked(event : ActionEvent) : void
+bilkaButtonClicked(event : ActionEvent) : void

### Attributes

**private Label nameLabel:** This attribute is the name label of the building.

**private Label infoLabel:** This attribute is the rent and buy information label of the building.

**private ImageView image:** This attribute is the image of the building.

**private Button buyButton:** This attribute is the buy button of the building.

**private Button okButton:** This attribute is the ok button that closes the pane.

**private Button s1Button:** This attribute is the buy button for 1 Starbucks of the building.

**private Button s2Button:** This attribute is the buy button for 2 Starbucks of the building.

**private Button s3Button:** This attribute is the buy button for 3 Starbucks of the building.

**private Button bilkaButton:** This attribute is the buy button for Bilka of the building.

**private Color colorSet:** This attribute is the color of the building.

**<u>Methods</u>**

**public void showPane():** This method shows the building pane.

**public void updateLabels():** This method updates the labels in the pane.

**public void showButtons(boolean isSet);** This method decides what buttons to show depending on whether the player is displaying one land or the land set.

**public void buyButtonClicked(ActionEvent event):** This method is the listener of the buy button.

**public void okButtonClicked(ActionEvent event):** This method is the listener of the ok button.

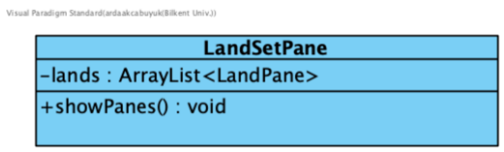
**public void s1ButtonClicked(ActionEvent event):** This method is the listener of the s1 button.

**public void s2ButtonClicked(ActionEvent event):** This method is the listener of the s2 button.

**public void s3ButtonClicked(ActionEvent event):** This method is the listener of the s3 button.

**public void bilkaButtonClicked(ActionEvent event):** This method is the listener of bilka buy button.

## LandSetPane Class


Visual Paradigm Standard(ardaakcabuyuk(Bilkent Univ.))
LandSetPane
-lands : ArrayList<LandPane>
+showPanes() : void

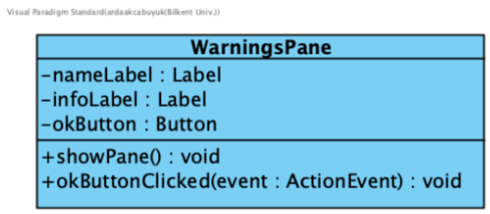
### Attributes

**private ArrayList<LandPane> lands:** This attribute is the array list of the LandPanes that belongs to the same color set. It allows us to display all color set at the same time.

### Methods

**public void showPanes():** This method displays the all color set.

## WarningsPane Class


Visual Paradigm Standard(ardaakcabuyuk(Bilkent Univ.))
WarningsPane
-nameLabel : Label
-infoLabel : Label
-okButton : Button
+showPane() : void
+okButtonClicked(event : ActionEvent) : void

### Attributes

**private Label nameLabel:** This attribute is the title of the warning pane.

**private Label infoLabel:** This attribute is the text info of the warning pane.

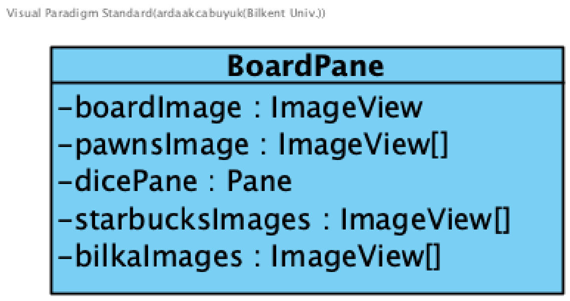
**private Button okButton:** This attribute is the ok button of the warning pane that closes the pane.

### Methods

**public void showPane():** This method displays the pane.

**public void okButtonClicked(ActionEvent event):** This method is the listener of the ok button.

## BoardPane Class


Visual Paradigm Standard(ardaakcabuyuk(Bilkent Univ.))
BoardPane
-boardImage : ImageView
-pawnsImage : ImageView[]
-dicePane : Pane
-starbucksImages : ImageView[]
-bilkaImages : ImageView[]

### Attributes

**private ImageView boardImage:** This attribute is the image of the game board.

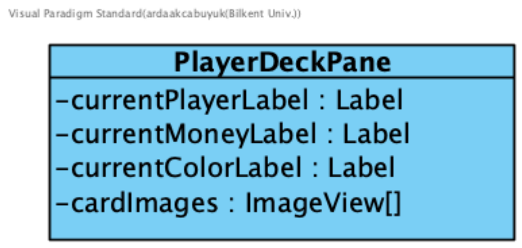
**private ImageView[] pawnsImage:** This attribute is the images of the Pawns on the game board.

**private Pane dicePane:** This attribute is the Pane in which the roll dice animation is shown.

**private ImageView[] starbucksImages:** This attribute is Starbucks images currently on the game board.

**private ImageView[] bilkaImages:** This attribute is Bilka images on the game board.

## PlayerDeckPane Class


Visual Paradigm Standard(ardaakcabuyuk(Bilkent Univ.))
PlayerDeckPane
-currentPlayerLabel : Label
-currentMoneyLabel : Label
-currentColorLabel : Label
-cardImages : ImageView[]

### Attributes

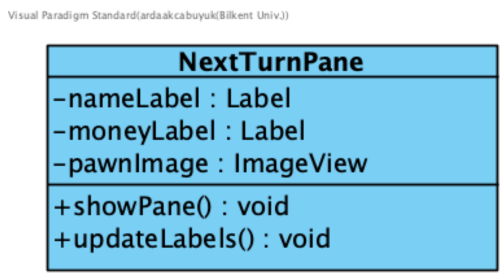
**private Label currentPlayerLabel:** This attribute is the name of the player who has the turn which will be displayed on the front credit card.

**private Label currentMoneyLabel:** This attribute is the balance of the player who has the turn which will be displayed on the front credit card.

**private Label currentColorLabel:** This attribute is the color of the player who has the turn which will be the border color of their credit card.

**private ImageView[] cardImages:** This attribute is the images of the credit cards that the players are represented with.

## NextTurnPane Class



### <u>Attributes</u>

**private Label nameLabel:** This attribute is the name of the Player that will play in the next turn.

**private Label moneyLabel:** This attribute is the balance of the Player that will play in the next turn.

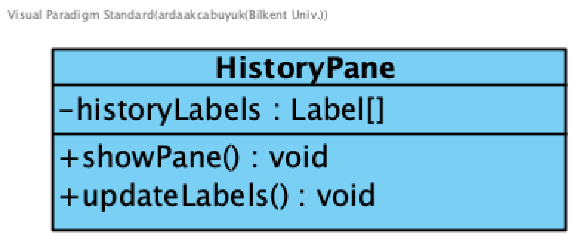
**private ImageView pawnImage:** This attribute is the Pawn image of the Player that will play in the next turn.

### <u>Methods</u>

**public void showPane():** This method shows the NextTurnPane.

**public void updateLabels():** This method is called in each turn, changing the player info on the pane. It updates the player that is displayed on this panel.

## HistoryPane Class



### Attributes

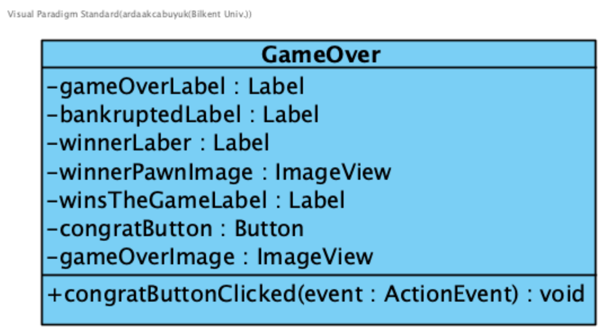
**private Label[] historyLabel:** This attribute keeps track of the events that take place within the game.

### Methods

**public void showPane():** This method shows the history pane.

**public void updateLabels():** This method updates the labels in the pane.

## GameOver Class


Visual Paradigm Standard(ardaakcabuyuk(Bilkent Univ.))
GameOver
-gameOverLabel : Label
-bankruptedLabel : Label
-winnerLaber : Label
-winnerPawnImage : ImageView
-winsTheGameLabel : Label
-congratButton : Button
-gameOverImage : ImageView
+congratButtonClicked(event : ActionEvent) : void

### Attributes

**private Label gameOverlabel:** This attribute is the game over text.

**private Label bankruptedLabel:** This attribute is “Everyone bankrupted” text.

**private Label winnerLabel:** This attribute is the winner player.

**private Label winsTheGameLabel:** This attribute is “Wins the game” text.

**private ImageView winnerPawnImage:** This attribute isa picture of the winner player’s pawn.

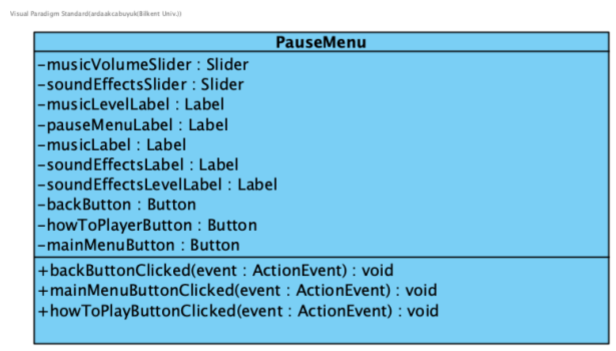
**private ImageView gameOverImage:** This attribute isa cash picture.

**private Button congratButton:** This attribute is the congrats button that closes the pane.

### Methods

**public void congratButtonClicked(ActionEvent event):** This method is the listener of the congratButton.

## PauseMenu Class


Visual Paradigm Standard(ardaakcabuyuk(Bilkent Univ.))
PauseMenu
-musicVolumeSlider : Slider
-soundEffectsSlider : Slider
-musicLevelLabel : Label
-pauseMenuLabel : Label
-musicLabel : Label
-soundEffectsLabel : Label
-soundEffectsLevelLabel : Label
-backButton : Button
-howToPlayerButton : Button
-mainMenuButton : Button
+backButtonClicked(event : ActionEvent) : void
+mainMenuButtonClicked(event : ActionEvent) : void
+howToPlayButtonClicked(event : ActionEvent) : void

### Attributes

**private Slider musicVolumeSlider:** This attribute is the music volume slider that allows players to adjust music level.

**private Slider soundEffectsSlider:** This attribute is the sound effects volume slider that allows players to adjust sound effects level.

**private Label musicLevelLabel:** This attribute shows the music slider level.

**private Label pauseMenuLabel:** This attribute is the title of the pane.

**private Label musicLabel:** This attribute is the name of the music slider.

**private Label soundEffectsLabel:** This attribute is the name of the sound effects slider.

**private Label soundEffectsLevelLabel:** This attribute shows the sound effects slider level.

**private Button backButton:** This attribute is the back button.

**private Button howToPlayButton:** This attribute is the how to play button.

**private  Button mainMenuButton:** This attribute is the main menu button.

**Methods**

**public void backButtonClicked(ActionEvent event):** This method is the listener of the back button that closes the Pause Menu.

**public void mainMenuButtonClicked(ActionEvent event):** This method is the listener of the Main Menu button that closes the game and goes back to the Main Menu.

**public void howToPlayButtonClicked(ActionEvent event):** This method is the listener of the How To Play button that opens How To Play pane during the game.

# 4. Low-level Design

## 4.1 Object Design Trade-offs

### 4.1.1 Efficiency v. Portability

The fact that the project can be used on more than one platform may adversely affect its efficiency on these platforms. Because the different requirements and operating styles of each platform may not be fully efficient with the Java language in which the project was written. However, such a trade-off was made, as the game was thought to be on multiple platforms and it would not feel much negativity in the user experience in terms of performance/efficiency.

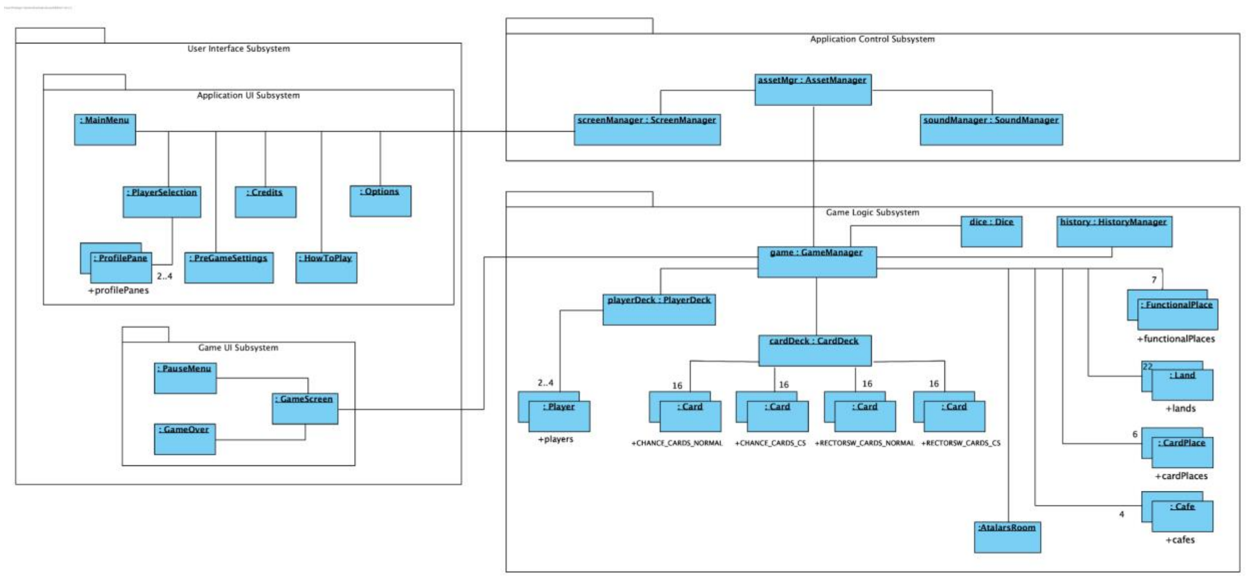
### 4.1.2 Cost v. Readability

In the design part of the project, care was taken to establish a more readable and clear hierarchy among classes and interfaces, although it is longer, more complex and more time consuming.

### 4.1.3 Lite development vs. Functionality

Due to the limited time to develop the game, it was decided not to add some features that would make the game more functional (more competitive and fun). As mentioned in the analysis report, some optional features can be added when the game is extended. However, we took care to postpone these features that do not affect the gameplay but only improve the gameplay in the rapid development process and create a game that can be played sufficiently. In the future versions, features such as chatroom and bargaining can be added to further increase the interaction between users.

## 4.2 Final Object Design



The final object design shows the connections between packages and the classes. The diagram includes the instances which will be created during the execution of the software. The attribute names are not included since they are explained in the above sections.

## 4.3 Packages

**javafx.*:** The JavaFX package is a GUI library that allows us to use JavaFX User Interface components such as Panes, Labels, Buttons, etc. It also allows us to create simple animations and 3D objects.

**java.io.*:** Input output library of Java.

**java.util.*:** Java library that allows us to use ArrayList and Scanner.

## 5. References

[1] JavaFX. [Online]. Available: https://openjfx.io/. [Accessed: 28-Nov-2020].

[2] “Package java.io,” *java.io (Java Platform SE 7 )*, 24-Jun-2020. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html. [Accessed: 28-Nov-2020].

[3] *What is Object Diagram?* [Online]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-object-diagram/. [Accessed: 28-Nov-2020].

[4] *UML: Modeling Software Architecture with Packages*. [Online]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/modeling-software-architecture-with-package/. [Accessed: 28-Nov-2020].