



**EÖTVÖS LORÁND TUDOMÁNYEGYETEM**

**INFORMATIKAI KAR**

**Információs Rendszerek Tanszék**

---

# **Blackjack kártyajáték implementálása**

Témavezető:

**Gombos Gergő**

adjunktus

Készítette:

**Faludi Péter**

programtervező informatikus BSc szak

Budapest, 2019

## Tartalom

Bevezetés.....	5
Felhasználói dokumentáció .....	6
Rendszerkövetelmények.....	6
Állományok .....	6
Adatbázis .....	6
Szerver indítása.....	9
Használat .....	9
Bejelentkezés .....	9
Regisztráció .....	11
Lobby .....	12
Játék menete .....	14
Kör vége és pontozás .....	17
Alkalmazásból való kilépés.....	18
Fejlesztői dokumentáció .....	19
A feladat specifikációja.....	19
Adatbázis diagram: .....	20
Az adattáblák .....	20
userdata tábla .....	20
player tábla .....	20
Fejlesztői környezet .....	21
Megoldási terv .....	21
Kommunikáció.....	21
Kliens oldal.....	21
Szerver oldal .....	22
Játék logika .....	22
A fejlesztés folyamata .....	24
Részletek .....	25
Szerver oldal uml-diagramja .....	25
Kliens oldal uml-diagramja .....	26
PanelBox-ból leszármazó panelek .....	27
A server.logic package .....	27
Suit és Rank osztály .....	27
Card osztály .....	28
Deck osztály .....	29
Shoe osztály .....	29
Hand osztály .....	30

RuleHand osztály .....	30
Player osztály .....	32
ServerTable osztály .....	35
A server.gameServer package .....	36
GameServer osztály .....	37
ServerListener osztály .....	39
Validate osztály .....	39
WatingRoom osztály .....	40
A server.chatServer .....	41
ChatServer és ChatServerListener osztály .....	41
ChatPartner osztály .....	41
A server package .....	42
BCrypt és BcryptHashing osztály .....	42
DatabaseInitalizer osztály .....	42
A dao package .....	43
DefaultDao<T extends Player> osztály .....	43
PlayerDao extends DefaultDao<Player> osztály .....	45
A view.panels .....	46
Absztrakt PanelBox extends VBox osztály .....	46
• BetBox .....	46
• BothBox .....	46
• ContinueBox .....	46
• DoubleDownBox .....	46
• HitStayBox .....	46
• InsuranceBox .....	46
• ChatBox osztály .....	47
• DealerBox osztály .....	47
• ChatPane osztály .....	47
• MessageSendBox osztály .....	47
• InfoBox osztály .....	47
• ControllerEmpty osztály .....	47
Table osztály és a TableView osztály .....	49
Lobby és a LobbyView osztály .....	52
PlayerBox osztály .....	53
A login package .....	55
Login osztály .....	55

IpController osztály.....	55
LoginFXMLcontroller osztály .....	56
RegisterFXMLController osztály .....	56
MessagePopUp osztály .....	57
A client package .....	57
TimeOut és TimeOutTask osztály .....	57
GUITimer osztály .....	57
CardFactory osztály .....	57
LobbyController osztály.....	58
ClientModel osztály .....	59
ClientController osztály .....	60
ChatThread és ChatModel osztály .....	61
Tesztelés.....	62
Bejelentkezés.....	62
Lobby .....	63
Játék .....	64
2 vagy több játékosnál.....	64
Chat .....	65
Fejlesztési lehetőségek.....	66
Hivatkozások.....	67

## Bevezetés

A szakdolgozat a Blackjack nevű kártyajáték megvalósítása a kliens-szerver modell alapján, ahol a szerver tölti be a bank(osztó) szerepét, míg a kliensek a játékosokét. A játékot egy hagyományos BlackJack asztalon játszik több 52 lapos francia paklival (Jokerek nélkül). A klienshez tartozik egy grafikus felület, ahol a játékasztal, valamint az eddig gyűjtött nyereményeket láthatjuk, a szerver csak információkat közöl a kliensekkel.

Egy játékot legfeljebb 4 játékos játszhat, akik külön-külön az osztó ellen játszanak, tehát nem egymás ellen. A játék előtt a kliensek csatlakoznak a szerverhez, ami után egy várakozó szobába kerülnek. A játék akkor kezdődik el, ha egy játékos(kliens) csatlakozik a szerverhez ezután választ egy elérhető játékasztalt. A játék során először minden játékos megkapja a pénzét, ezután megteszik a játékosok a tétjeiket ezt követően a szerver minden játékosnak és saját magának is oszt egy lapot színével felfelé és oszt ugyanilyen módon még egy kört a játékosoknak de ebbe a körbe saját magának a már színnel lefelé oszt lapot így a játékosok nem tudják, hogy milyen a 2.lapja az osztónak. A játékos úgy nyerhet, ha az ő lapjai közelebb vannak összértékben 21-hez mint az osztójé viszont azt nem haladják meg. A játékos addig kérhet lapot, amíg nem éri el a 21-et ha túllépi akkor a játékos vesztett az osztó csak ezután kérhet lapot ugyanezen a szabályok mentén. (A pontos játékszabály: [\[1\]](#)) Amennyiben a játékos nyert akkor a kivételes esetektől eltekintve a feltett tét kétszeresét nyeri, ha az osztó nyert akkor a játékos elbukta a feltett tétet, ami a bankhoz kerül.

Egy játékos mindaddig folytatja a játékot ameddig el nem fogy a pénze, ki nem lép a játékból vagy bezárja az alkalmazást, ebben az esetben a játék folytatódik, ha tétrakás előtt lépett ki a játékos akkor a minimális tétet rakja fel a szerver a nevében, illetve mindig a megállást választja a játékos helyett. Amennyiben egy játékos kilépett a körvégen automatikusan lekerül az asztalról, ekkor több játékos esetén a többi játékos sem fogja tovább látni. A játékos a játékok között láthatja az eredménytáblát. A játékot Java nyelven fogom megvalósítani. A játékosok és a játékok eredményei MySQL adatbázisban fognak tárolódni. A kliensek a szerverrel socket-eken keresztül kommunikálnak, szöveges üzenetek formájában. A játék grafikus felülete JavaFx nyelven íródott.

# Felhasználói dokumentáció

## Rendszerkövetelmények

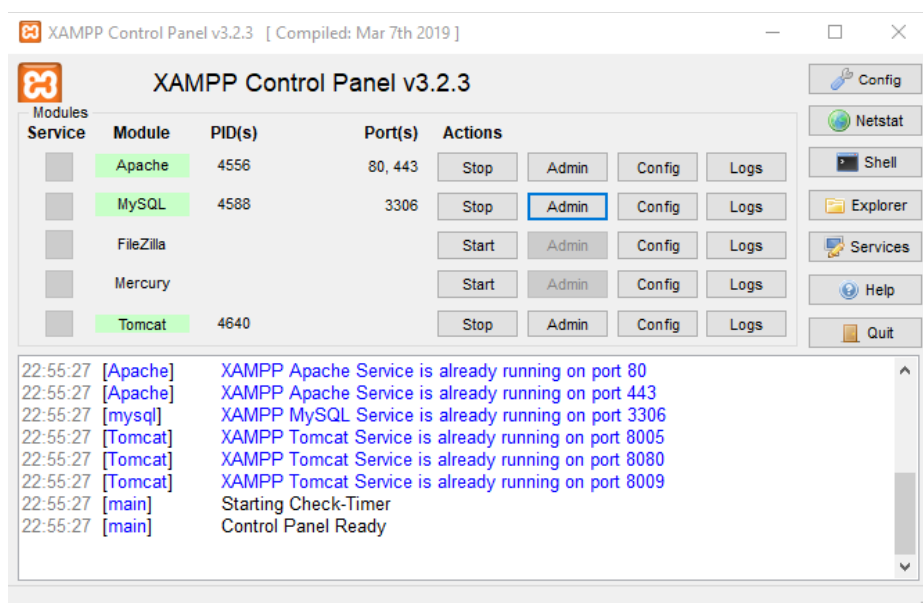
A játék platform független Java nyelven íródott. A játék futtatásához szükséges legalább Java SE Runtime Environment 8. A játék(kliens) futtatásához, nincs szükség telepítésre csak a futtatható állományra illetve, hogy csatlakozzon a szerverhez, aminek az elérési címe a játék elején beállítható (alapértelmezetten localhoston a 4444 illetve 4445-ös port), a szerver futtatásához szükséges egy blackjack mySQL adatbázis a nbuser,nbuser felhasználónév-jelszó párossal (localhost:3306/blackjack) táblákat a DatabaseInit állomány hozza létre. Hardver követelményei minimálisak. Ajánlott képernyő felbontás minimum 1366x768.

## Állományok

- Szerver.jar
- Kliens.jar
- DatabaseInit.jar

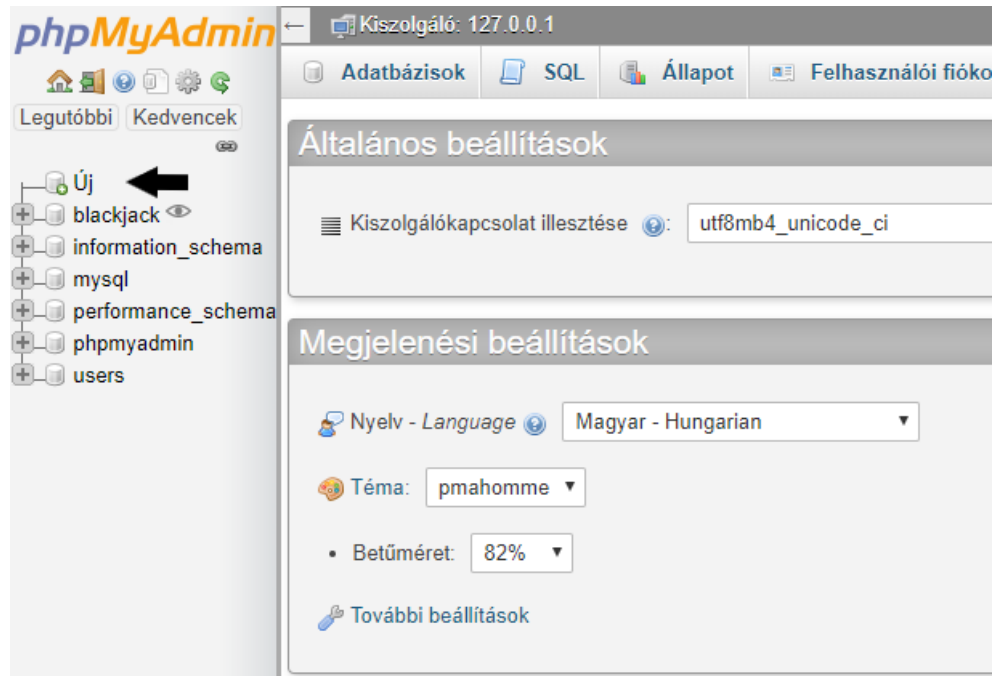
## Adatbázis

Az adatbázis létrehozását XAMPP webservert-szoftver csomaggal ajánlom [5], illetve a weboldalon található egy telepítési útmutató is. Ha az előbbi szoftvert választjuk az adatbázis telepítésére, akkor az útmutató szerint telepítsük a programot, ezután nyissuk meg a programot és indítsuk el az Apache,MySQL és Tomcat service-öket.



1. ábra

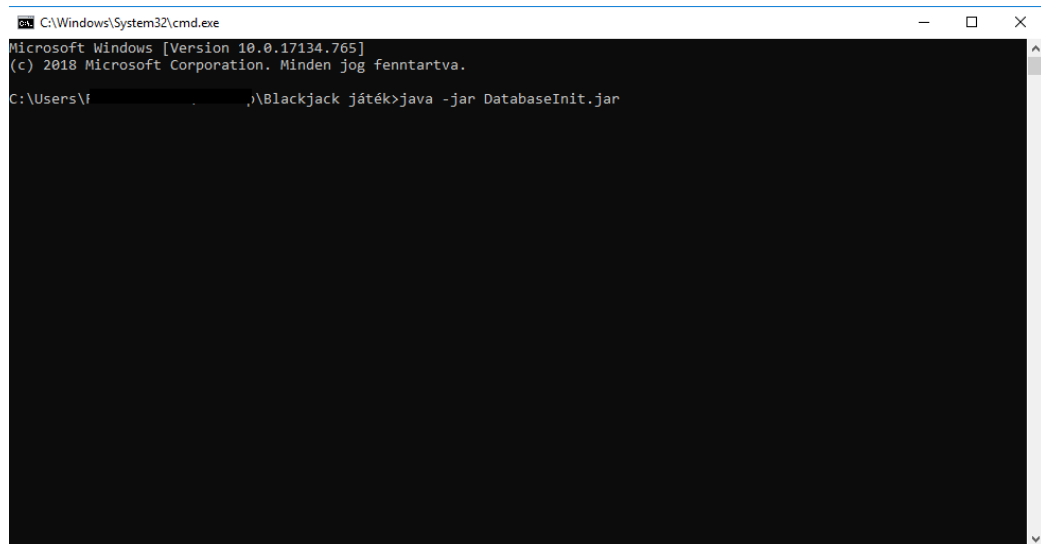
Ezután kattintsunk a MySQL admin gombjára ezt követően egy webes felületen fogja megjeleníteni a kezelőfelületet.



2. ábra

A weboldal baloldalán található listából kattintsunk az Új pontra ,majd a megjelenített ablakban az Adatbázis neve helyére írjuk be, hogy blackjack és kattintsunk a létrehozásra

Ha már létrehoztunk egy adatbázist a localhost:3306 címen, szerver számára (Pl XAMMP webservert-szoftver csomaggal), akkor futtassuk parancssorból a a DatabaseInit programot , ami legenerálja az adatbázis tábláit illetve létrehoz 4 teszt felhasználót.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.17134.765]
(c) 2018 Microsoft Corporation. Minden jog fenntartva.
C:\Users\F... \Blackjack játék>java -jar DatabaseInit.jar
```

3. ábra

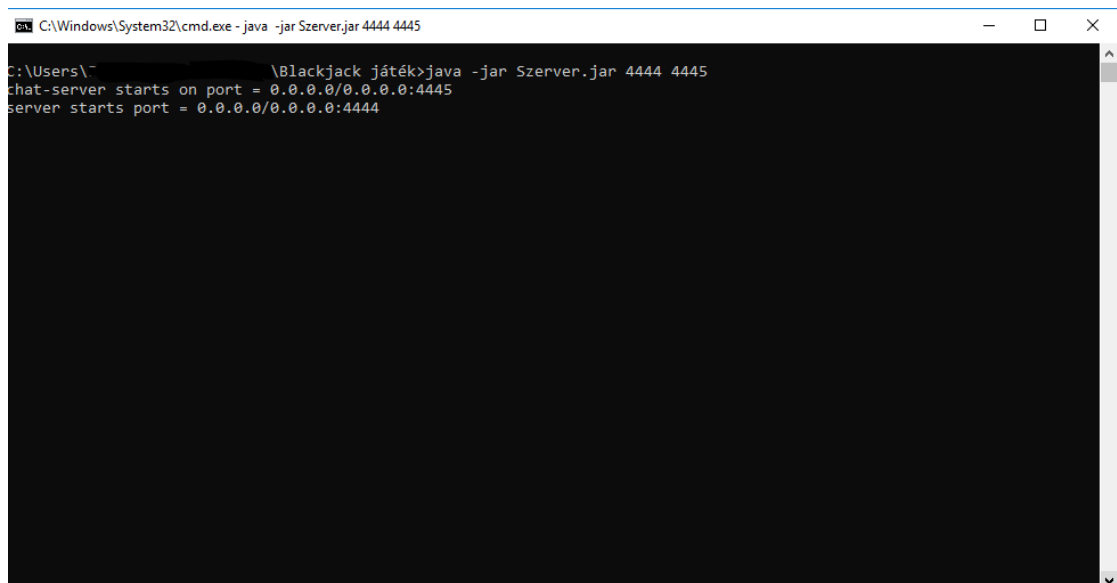
(Későbbiekben az adatbázisban módosíthatók a játékosok adatai ezen a felületen). A  
teszt játékosok adatai :

Felhasználónév	Jelszó
test1	test1
test2	test2
test3	test3
test4	test4



## Szerver indítása

A szerver indítása parancssorból történik, ahol paraméterben kell megadni a szerver két portját vagy paraméter nélkül indítani ekkor a 4444 illetve 4445 porton fog elindulni. (Pl java -jar Szerver.jar 1234 1234, vagy java -jar Szerver.jar). A szerver a parancssorba a Ctrl+C billentyűparancs bevitelével lehet leállítani.

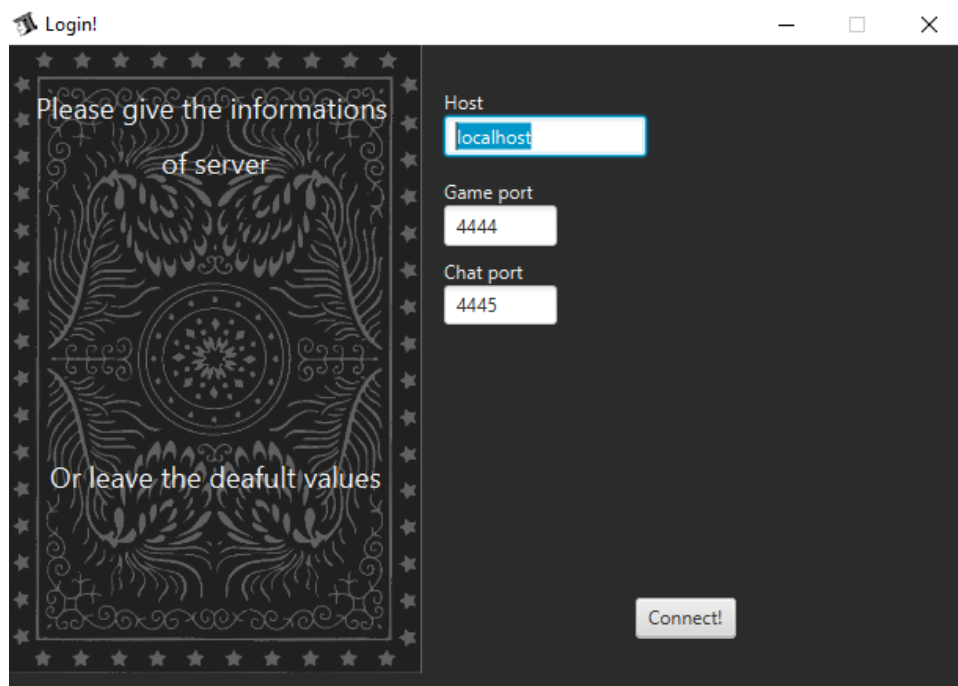
A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\System32\cmd.exe - java -jar Szerver.jar 4444 4445'. The command prompt shows the user's current directory as 'C:\Users\~' and the command '\Blackjack játék>java -jar Szerver.jar 4444 4445'. The output of the command is displayed on two lines: 'chat-server starts on port = 0.0.0.0/0.0.0.0:4445' and 'server starts port = 0.0.0.0/0.0.0.0:4444'. The rest of the command prompt window is empty.

4. ábra

## Használat

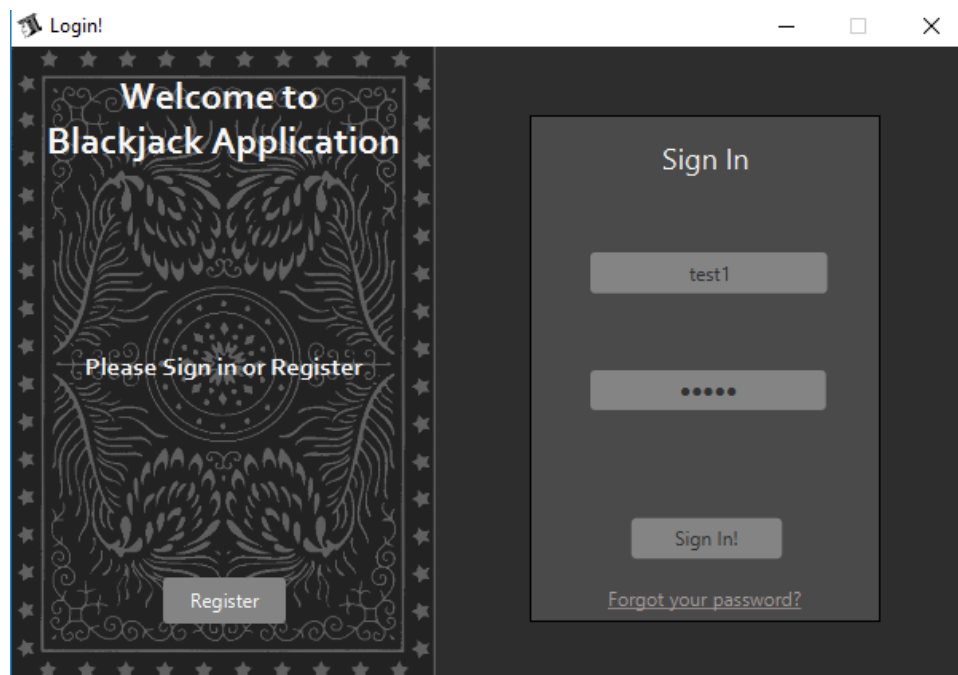
### Bejelentkezés

A játék futtatása történhet parancssori futtattassál, ez esetben a célhelyen kell egy parancssort megnyitni és következő parancsot begépelni: java -jar Kliens.jar. A játék futtatható ezenfelül a szokásos módon, az alkalmazásra való dupla kattintással. A játékos a program futtatása után egy bejelentkező ablakot fog látni, ahol megadhatja a szerver elérhetőségét.



5. ábra

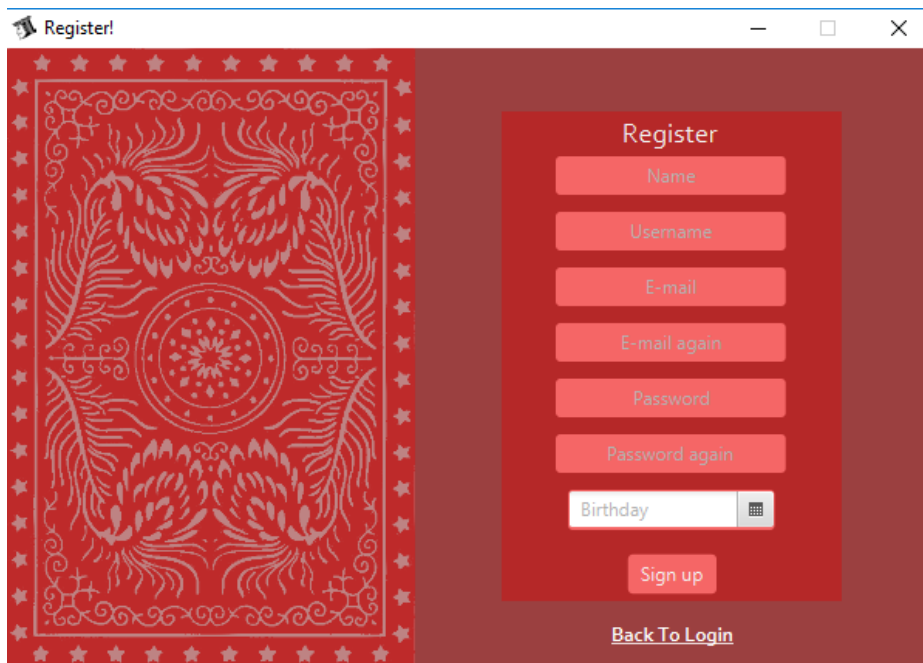
Ha a szerver nem elérhető a megadott címen a felhasználónak felugrik egy hibaüzenet, ha helyes a felhasználó tovább lép a bejelentkezési ablakra, ahol bejelentkezhet az adataival vagy regisztrálhat (Register gombra kattintva) a szükséges adatok megadásával. Ha a felhasználó hibás adatokat ad meg bejelentkezésnél, akkor hibaüzenet kap a rendszertől helytelen felhasználónév, illetve helytelen jelszó esetén .



6. ábra

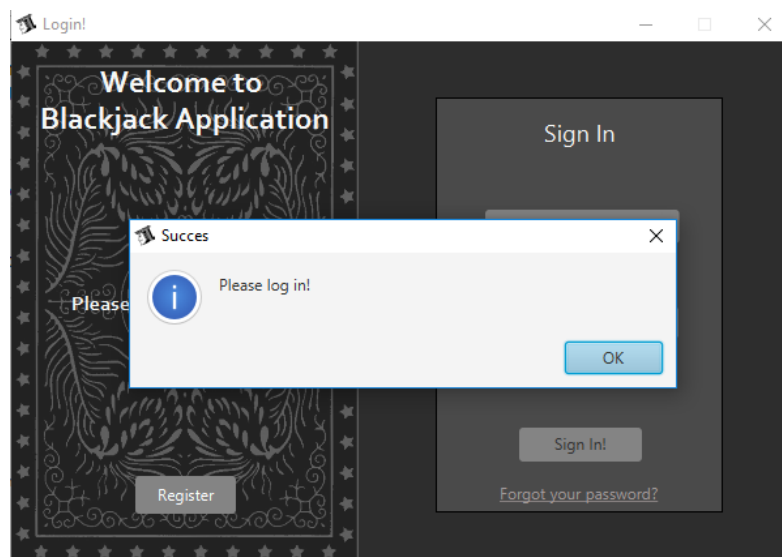
## Regisztráció

A Regisztráció során a felhasználónak meg kell adnia a nevét, egy felhasználónevet, egy e-mail címet, egy jelszót, illetve a születési dátumát. A program kétszer kéri be az e-mail címet, illetve a jelszót az elgépelés elkerülése végett, ha nem egyezik a két mező a rendszer figyelmezteti a felhasználót az eltérésre.

The image shows a web browser window with a title bar that says "Register!". The page has a dark red background. On the left side, there is a large, intricate white pattern resembling a traditional rug or tapestry. On the right side, there is a registration form with a light red background. The form contains the following fields and elements: a "Name" input field, a "Username" input field, an "E-mail" input field, an "E-mail again" input field, a "Password" input field, a "Password again" input field, a "Birthday" input field with a calendar icon, and a "Sign up" button. Below the "Sign up" button, there is a link that says "Back To Login".

7. ábra

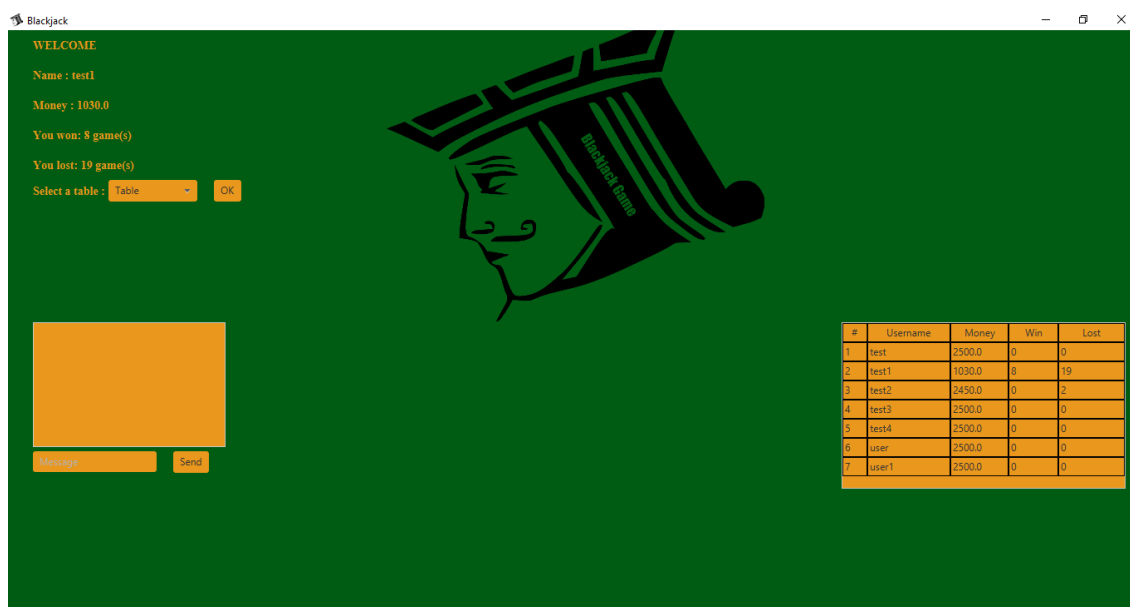
Az összes adat kitöltése kötelező, ha a felhasználó kihagyna egy mezőt erre is figyelmezteti a program. Ezen felül az e-mail címnek és a felhasználónévnek egyedinek kell lenniük, azok még nem szerepelhetnek az adatbázisban, ha mégis szerepelnek a program felhívja a felhasználó figyelmét erre és egy hibaüzenetet küld, a javítandó mezőről. Ha a regisztráció sikeres volt, akkor a felhasználó kap egy üzenetet, ezután a bejelentkező felületre kerül, ahol az új felhasználói fiókjával bejelentkezhet minden játékos kezdőtőkéje: 2500\$.



8. ábra

## Lobby

A felhasználó sikeres bejelentkezés után a Lobby-ba kerül ,



9. ábra

ahol láthatja a saját adatait, chat-elhet a többi játékosal, megnézheti a játékosok eredményét, illetve csatlakozhat egy asztalhoz játék céljából.

**WELCOME**

**Name : test1**

**Money : 1030.0**

**You won: 8 game(s)**

**You lost: 19 game(s)**

Select a table : Table ▾ OK

1. min:10\$  
 2. min:50\$  
 3. min:100\$  
 4. min:200\$

10. ábra

A játékosok eredményét az oszlopnévre kattintva tudja rendezni, csak olyan asztalhoz csatlakozhat, ahol a minimális tét kisebb, mint a játékos pénze.

#	Username	Money	Win	Lost
1	test1	50.0	5	14
2	test2	2450.0	0	2
3	test3	2500.0	0	0
4	test4	2500.0	0	0
5	test	2500.0	0	0

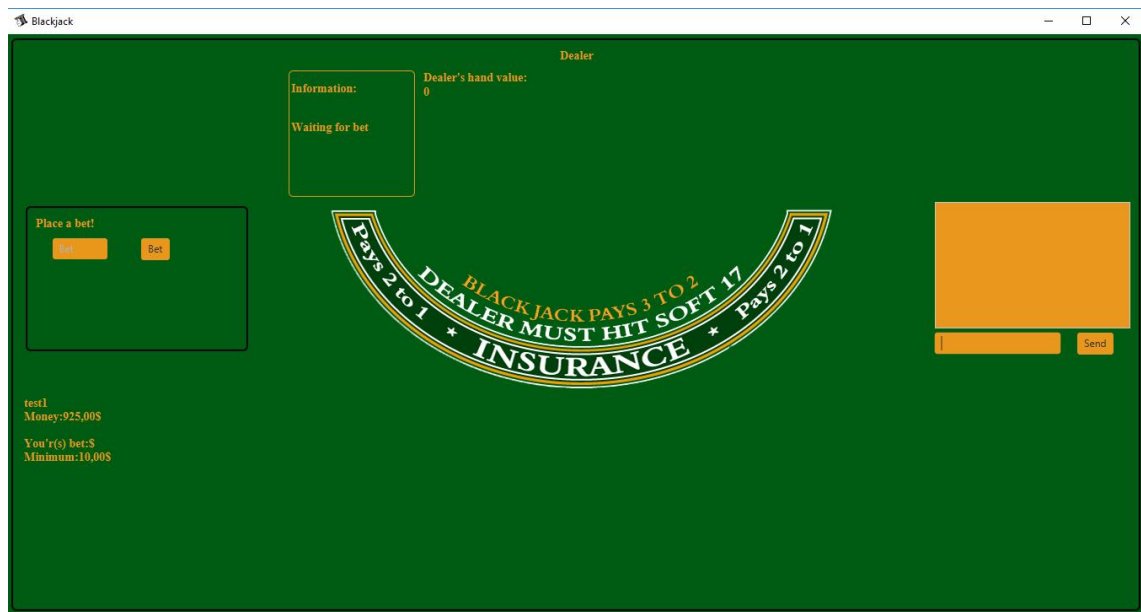
test2:Ez egy teszt.  
 test1:Fogadtam a tesztet

Message Send

11. ábra

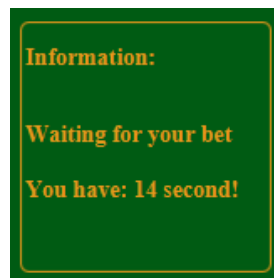
Egy játékos akkor kerül a játéktérre, ha az asztalon vége van az előző körnek, ha a felhasználó várakozik egy kör végére addig nem választhat másik asztal csak akkor, ha a kiválasztott asztalon végez a játékkal, ahonnét visszakerül a Lobby-ba és újra választhat asztalt.

## Játék menete



12. ábra

A játékot egy, vagy legfeljebb 4 játékos játssza, illetve a szerver. A játékost egy információs panelben értesíti a játék fejleményeiről ez a panel az osztó lapjai előtt jelenik meg.



13. ábra

Több játékos esetén a játékos saját panele a csatlakozás sorrendje szerint jelenik meg.



14. ábra

Először minden játékos megteszi a tétjeit, aminek minden asztalnál nagyobbbnak kell lennie a minimum tétnél.



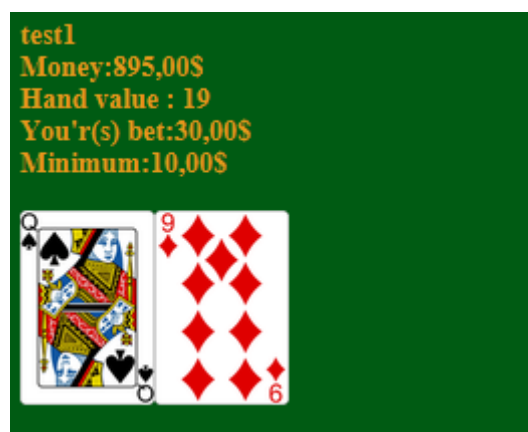
15. ábra

Egy játékos esetén a szerver a játékosnak oszt egy lapot, több játékos esetén minden játékosnak oszt egy lapot, ezután magának is oszt egy lapot színnel felfelé ezután megismétlődik az előző kör annyi változással, hogy az osztó magának egy kártyát oszt színnel lefelé.



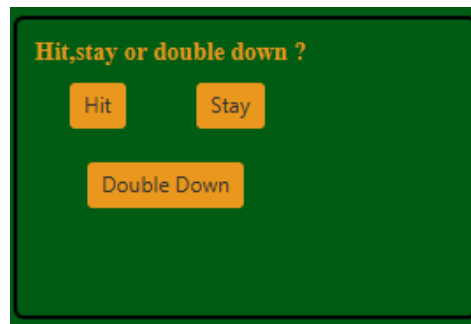
16. ábra

A játék ezután kezdődik, ha valamelyik játékos lapjainak értéke 21 (blackjack), akkor nyert (kivéve, ha az osztónak is 21-e van ekkor döntetlen és visszakapja a feltett értéket) ilyenkor a megtett tét 1,5 szeresét kapja meg.



17. ábra

Ha a játékosnak a lapjainak értéke nem 21, akkor 3 választása van (+1 speciális eset) választásra 15 másodperc árendelkezésre, amit az információs panelben egy számláló mutat a játékosnak:



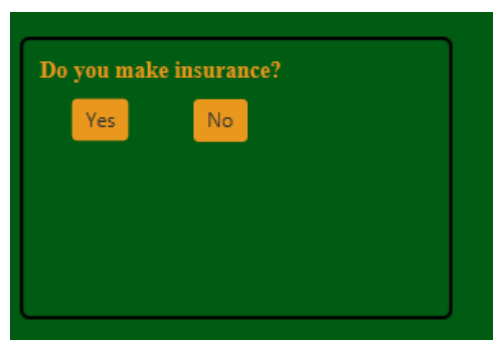
18. ábra

- dupláz ekkor a feltett pénzét megduplázza és ezután már csak egy lapot kap;
- lapot kér ezt a lépését addig ismétli, ameddig meg nem áll vagy lapjai értéke meg nem haladja a 21-et;
- illetve megállhat ekkor a következő játékos vagy osztó következik.



19. ábra

+1 eset: Ha az osztónak az első lapja ász akkor a játékos biztosítást köthet az osztó következő lapjára, ez a feltett tét fele.



20. ábra



### Kör vége és pontozás

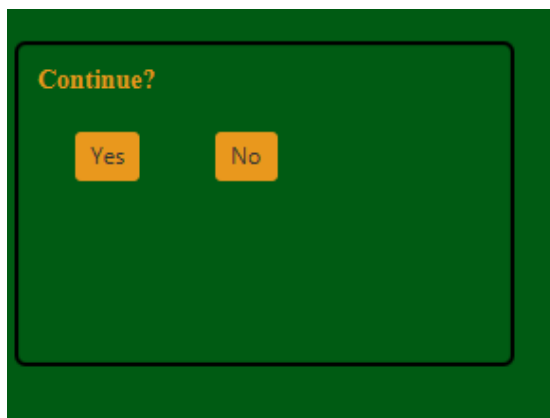
A játékos akkor nyer, ha lapjai értéke nem haladják meg a 21-et illetve az osztó lapjainak értéke vagy meghaladják a 21-et, vagy a játékos lapjainak értéke közelebb vannak a 21-hez.

A kifizetés a következőképpen zajlik:

- Ha a játékos normál módon nyer 1:1 arányban kap pénzt a téthez viszonyítva
- Ha a játékos első két lapjának értéke 21, akkor 3:2 arányban
- Ha duplázott, akkor a tétet duplázta és az első esett érvényes rá a továbbiakban
- Ha a biztosítást kötött 1:1 arányban kap pénzt a biztosítási téthez viszonyítva, ha az osztónak Blackjackje van.

Ha a játékos veszít, akkor a feltett tétet elveszíti.

Ezután a játékos eldöntheti, hogy szeretne tovább játszani.



21. ábra

Ha nem kívánja folytatni a játékot, akkor a játékos átkerül a Lobby-ba. Ha a játékos folytatja a játékot, akkor a várakozó játékosok csatlakoznak és megjelennek a játékos grafikus felületén és a játék folytatódik. (Több játékos esetén a távozó játékos panele eltűnik a kezelőfelületről.)



22. ábra

### Alkalmazásból való kilépés

A felhasználó bármikor bezárhatja az alkalmazást az ablak bezárásával, viszont, ha ezt a játékasztalon teszi, akkor a szerver kénytelen lejátszani helyette a játékot, aminél nagy eséllyel veszít. Ajánlott a Lobby-ba bezárni az alkalmazást a játék elhagyása után, ekkor a játékos statisztikái változatlanok maradnak.

# Fejlesztői dokumentáció

## A feladat specifikációja

A programnál a megvalósítandó cél egy grafikus felülettel rendelkező, többszemélyes, kliens-szerver alapú Blackjack kártyajáték implementálása volt. A játéknak lehetővé kellett tennie, hogy egy vagy több ember tudjon játszani a programmal, akár párhuzamosan több asztalnál is. A program létrehozásánál a grafikus felületet próbáltam, úgy megalkotni, hogy minden képernyő típus mellett használható legyen és felhasználó barát legyen. Az információ közlése automatikus és könnyen értelmezhető. Az osztó implementálásánál fontos volt, hogy a valós játékszabályok szerint játsszon. A játékosoknál egy-két speciális esetet kihagytam a megjelenítés miatt vagy, mert nem általánosan elfogadott lépés. A feladat elkészítése során az alap játék került implementálásra (split hand, surrender kivételével). Fontos szempont volt, hogy a játék több asztalon is fusson, ezáltal több játékos részt tudjon venni a játékban.

Elvárt funkciók:

Felhasználók kezelése:

- Játékosok adatainak tárolása adatbázisban
- Játékosok adatainak automatikus frissítése
- Játékosok jelszavainak biztonságos tárolása
- Játékosok bejelentkezésének a kezelése

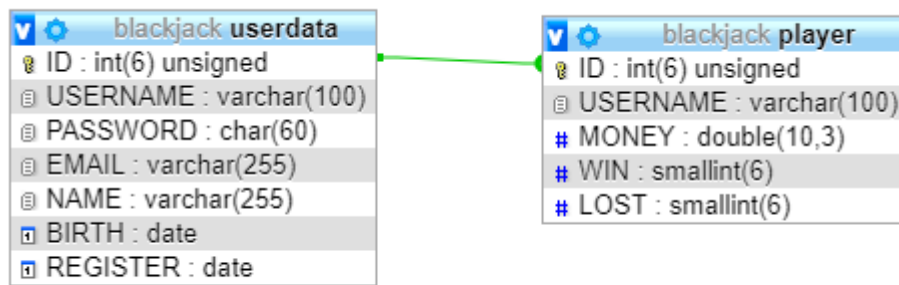
Szabályok megvalósítása:

- Eredeti játékszabályok betartása
- Eredetivel megegyező kompenzációs rendszer

Irányítás, megjelenítés:

- Egyszerű, egyértelmű felület
- Fontos információk automatikus közlése
- Egyszerű tovább lépési opciók
- Egyszerű egérrel és billentyűzettel való vezérlés

## Adatbázis diagram:



23. ábra

## Az adattáblák

Az adatbázisban 2 tábla van: a userdata tábla és a player tábla. A userdata tárolja a felhasználó által megadott adatokat, a player tábla tárolja játék során használt adatokat.

### userdata tábla

- ID: az elsődleges nem nulla azonosító, egy egész számot tartalmazó azonosító
- USERNAME: a felhasználó felhasználónevét tároló mező (szöveges formátum)
- PASSWORD: a felhasználó jelszavát salted hash-ben tároló mező (szöveges mező)
- EMAIL: a felhasználó e-mail címét tároló mező (szöveges mező)
- BIRTH: a felhasználó születési dátumát tároló mező (dátum formátum)
- REGISTER: a regisztráció dátumát tároló mező (dátum formátum)

### player tábla

- ID: az elsődleges nem nulla azonosító, egy egész számot tartalmazó azonosító
- USERNAME: a felhasználó felhasználónevét tároló mező (szöveges formátum)
- MONEY: a felhasználó pénz összegét tároló mező (double típus)
- WIN: a felhasználó által megnyert játékok száma (egész típus)
- LOSE: a felhasználó által elveszített játékok száma (egész típus)

## Fejlesztői környezet

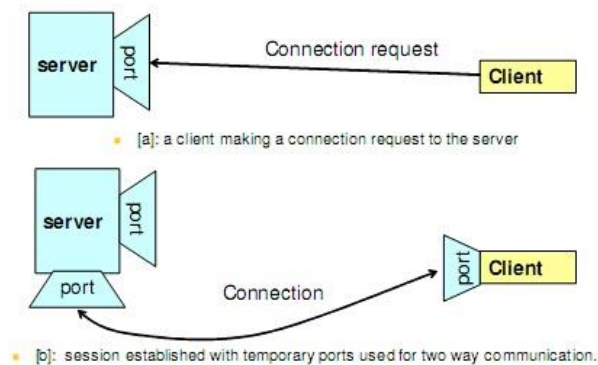
A program Windows 10 alatt íródott, Java nyelven, a grafikus felület játékhoz tartozó része JavaFx-ben íródott, a bejelentkezési felület Fxml-ben Scene Builder segítségével [11], NetBeans compiler-rel. Az adatbázist XAMMP szoftver segítségével tudtam vizsgálni.

## Megoldási terv

A játék megvalósításának a megkezdésénél két részre bontottam a programot, a szerver részre, illetve a kliens részre. Ezután azokat kisebb egységekre bontottam, amiket egyes package-ek szimbolizálnak. A játék implementálása a játékhoz elengedhetetlen típusok létrehozásával kezdődött. A megvalósításnál fontos szempont volt a modularitás, mind a funkcionális, mind a megjelenítésért felelős osztályoknál.

## Kommunikáció

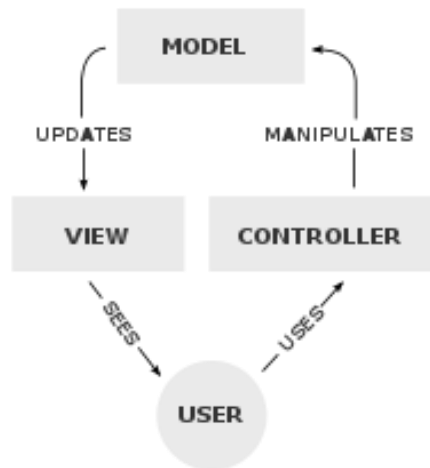
A program socketeken kommunikál. A kommunikáció karakterláncokkal történik, amit kisebb karakter láncokra bont a kliens és ezeket a komponenseket a vezérlőosztály értelmez egy nagyobb switchen keresztül és ez alapján frissíti a model-osztályt, illetve ezt átadja a grafikus felületnek megjelenítésre.



24. ábra

## Kliens oldal

A kliens oldal az MVC-architektúra alapján íródott, így szét válik a megjelenítés, az adattárolás és a vezérlés. Egymástól elkülönülve, egymást értesítve működnek a különböző komponensek. [5]



25. ábra

## Szerver oldal

A szerver oldal megvalósítása során a modularitás megteremtése volt a cél. A játék logikáját részegységekre bontva, külön egy asztalra bontottam és asztal viselkedését a játékosok lépéseitől tettem függővé. A szerver implementációja egy egyszerű server-socket felkészítve több kliens fogadására, amit kiegészítettem egy validációs illetve egy várakozó osztállyal. A szerver adattárolása egy mySQL adatbázisban történik, amihez egy Data Access Object-tel fér hozzá a szerver. A DAO egy JDBC connector-ral csatlakozik az adatbázishoz.



26. ábra

## Játék logika

A játék logikája teljes mértékben a Blackjack szabályaira támaszkodik. Mivel a játékos csak és kizárólag az osztó ellen játszik, ezért csak az osztó viselkedését kell szimulálnunk, viszont az osztó szabályok szerint játszik ezért nekünk az a célunk, hogy ezt implementáljuk. Ha az osztó lapjainak értéke meghaladja a 17-et akkor az osztó megáll egyébként lapot kér (ha soft-hand-je van és meghaladja a 21-et akkor levonunk 10-et az értékből (soft hand ha van a kezében egy Ász)). Ezen felül arra figyelünk, hogy a játékos mikor veszít, illetve nyer, ahogy a bevezetésben már részleteztük. A játék logikája leegyszerűsítve: egy Countdown latch segítségével megvárjuk az összes tétet, végig iterálunk a játékosoknak és osztóunk egy lapot, ezután az osztó is kap egy lapot és újra

végrehajtjuk ezt, de az osztó 2.lapjának most a hátoldalát jelenítjük meg. Ezután egy olyan iteráció következik, ahol a játékos addig játszik amíg meg nem áll vagy veszít, esetleg dupláz ekkor még egy lapot kap és duplázódik a tétje (ha az osztó első lapja ász biztosítást köthet), ezután folytatjuk a következő játékkal az iterációt ugyanezzel a sorrenddel...így tovább az utolsó játékosig ezután az osztó következik a fentebb említett logika alapján. Ezután még egy iteráció következik, aminél ellenőrizzük, hogy ki szeretné folytatni a játékot és a bent maradó játékosok ezt a logikát követve játszanak tovább. Ha valaki kilép a játék közben a szerver úgy játszik, hogy a minimális tétet teszi meg helyette és ha rá kerül a sor megáll, így valószínűleg veszít a játékos, de akár nyerhet is, a kör végén eltávolításra kerül az asztalról. A játékosnak minden választásra 15 másodperce van, ha letelik a 15 másodperc, akkor a kilépéskor használt lépések hajtódnak végre. A játék kommunikációja socketeken történik, ahol szöveges kommunikáció folyik, a szerveren fut a játék a kientől csak az egyes lépések opcióit várja. A kliens grafikus felülete a szöveges üzenet tartalma szerint változik.

A szerver oldal:

- A server.logic package tartalmazza a játék logikáját és a hozzá tartozó típusok implementációját.
- A server.gameServer package tartalmazza szerver azon részét, amely biztosítja a játékosok csatlakozását és a játékon kívüli kommunikációt illetve a socketek kezelését.
- A server.chatServer package chat-szerver implementációja.
- A server package-ben található BCrypt osztályok a jelszó salted hash-ét előállító illetve ellenőrző osztályai, illetve az adatbázis tábláinak létrehozásáért felelős osztály. (Nem saját osztályok)
- A dao package (Data Access Object) felel a szerver és az adatbázis kapcsolataért és az adatok kezeléséért.

A kliens oldal:

- A client package itt található a játékos számára a játék és a chat logikáját tartalmazó osztályok, illetve a modellek.
- A view tartalmazza az asztal, illetve a lobby felültért felelős osztályokat
- A view.panels package a megjelenítésnél használt képernyő paneleket tartalmazza

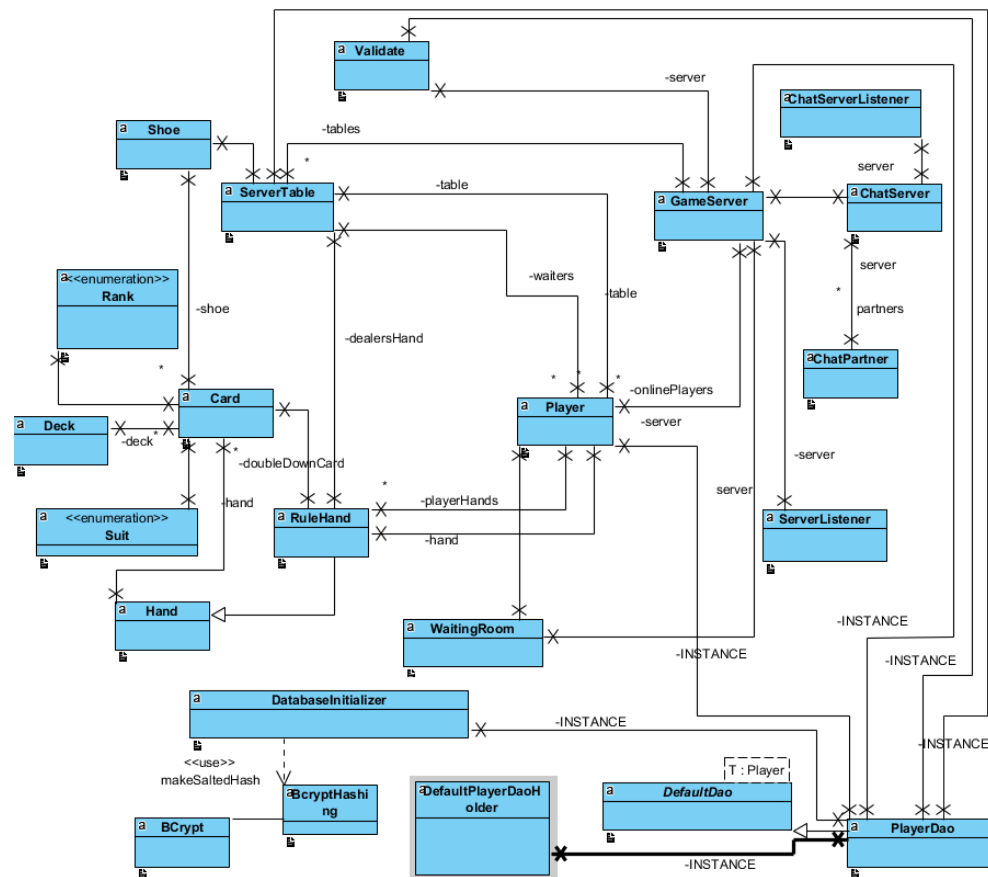
## A fejlesztés folyamata

A kliens oldal tervezésekor az MVC-architektúrához próbáltam tartani magam és ez alapján kezdtem neki a kód megírásának. A kód megírásának az első fázisában arra törekedtem, hogy a szerver oldalon az összes olyan osztályt megírjam, ami a játék során használt objektumokat leírják (kártya, pakli, shoe, asztal, játékos) ezt követően a játék logikáját leíró osztályokat készítettem el. A kód írás első célja az volt, hogy a legalapvetőbb játékfunkciók működjenek és kommunikáljon egymással a szerver és egy kliens program. Az első fázis megvalósítása közben a próbáltam sok módszert alkalmazni a kommunikációra, de végül a szöveges változat mellett döntöttem mert így nem kellett semmilyen megkötést se figyelembe venni. A szerver oldalt felkészítettem több kliens kiszolgálására. A második fázis volt a felhasználók kezelése. A mySQL adatbázis mellett azért döntöttem, mert könnyű hozzáférést biztosít az adatokhoz és sok szoftver biztosít hozzá grafikus felületet, ami a tesztelésnél és a tervezésnél megkönnyítette a dolgom. Azért esett a választásom a két adattáblára, mert azt szerettem volna, hogy a játéknak lenne egy külön csak a neki fenttartott tábla, amit egyszerűen módosíthat míg a másik tábla a játék számára irreleváns adatokat, személyes adatokat tárol. Ezután létrehoztam FXML-ben 2 ablakot, amik az adatbevitelt szolgálták és megírtam a hozzájuk tartozó controller osztályokat. Ezt követően a szerver oldalt is fel kellett készíteni az adatok fogadására, illetve azoknak a tárolására. A szerver oldalon először létrehoztam egy DAO-t és ehhez írtam egy validáló osztályt, ami a DAO-n keresztül lekéri a szükséges adatok a bejelentkezéshez és ellenőrzi az adatok helyeségét és értesíti a felhasználót. Amikor az adatbázisba felkerültek már az első adatok, akkor a következő lépés az volt, hogy hogy tároljam el a jelszavakat, mert azokat bárki láthatta az adatbázisban. Ekkor találtam rá a Salted Hash nevű titkosításra, ami biztosítja, hogy a tárolt karakterláncok alapján az eredeti jelszó ne legyen visszafejthető. A program grafikus megjelenítéséhez először a Java Swinget szerettem volna alkalmazni, viszont elég hamar kijöttek a Swing hátrányai, mindent előre definiálni kellett és az eszköztára véleményem szerint idejét múlt. Miután a Swing nem volt számomra alkalmas, ezért a JavaFx-re esett a választásom, ami nagyobb mozgásteret engedett és dinamikusan változó kezelő felületet lehet vele létrehozni. Elkülönítve kezdtem el írni a kezelőfelületet, ami így jobban tesztelhető volt és gyorsabb is volt a fejlesztése. A grafikus felület kidolgozásánál próbáltam minél kisebb modulokra szétbontani az asztalt, így jobban tudtam szétválasztani a későbbi funkciókat, szempont volt, hogy egy panel csak egy logikai egységet reprezentáljon (például tétrakás, lap kérés...). A fejlesztés legnehezebb része az volt, hogy a meglévő komponenseimet összeillesszem és azoknak a működését szinkronba hozzam. Ezután elvégeztem a szükséges kiegészítéseket a programon, aminél figyelembe vettem, hogy egy felhasználó miként viselkedhet és az ezekből fakadó hiba lehetőségeket próbáltam kiküszöbölni. Ennek a folyamat során az volt a célom, hogy a játékos ne tudjon többször bejelentkezni ugyanazzal a fiókkal egy időben, illetve, ha játék közben kilépne akkor a szerver ezt kezelje és folytassa a játékot, ezen felül az alkalmazás kapott egy időzítőt, így, ha valamelyik játékos elhagyná a számítógépet játék közben a szerver lejátssza helyette a játékot és a kör végén leveszi az asztalról. A tesztelés során előjöttek kisebb hibák, ezeket sikerült kisebb módosításokkal kiküszöbölni. A fejlesztés végén még bekerült egy logfile írás funkció minden asztalnak külön logfile-ja van.



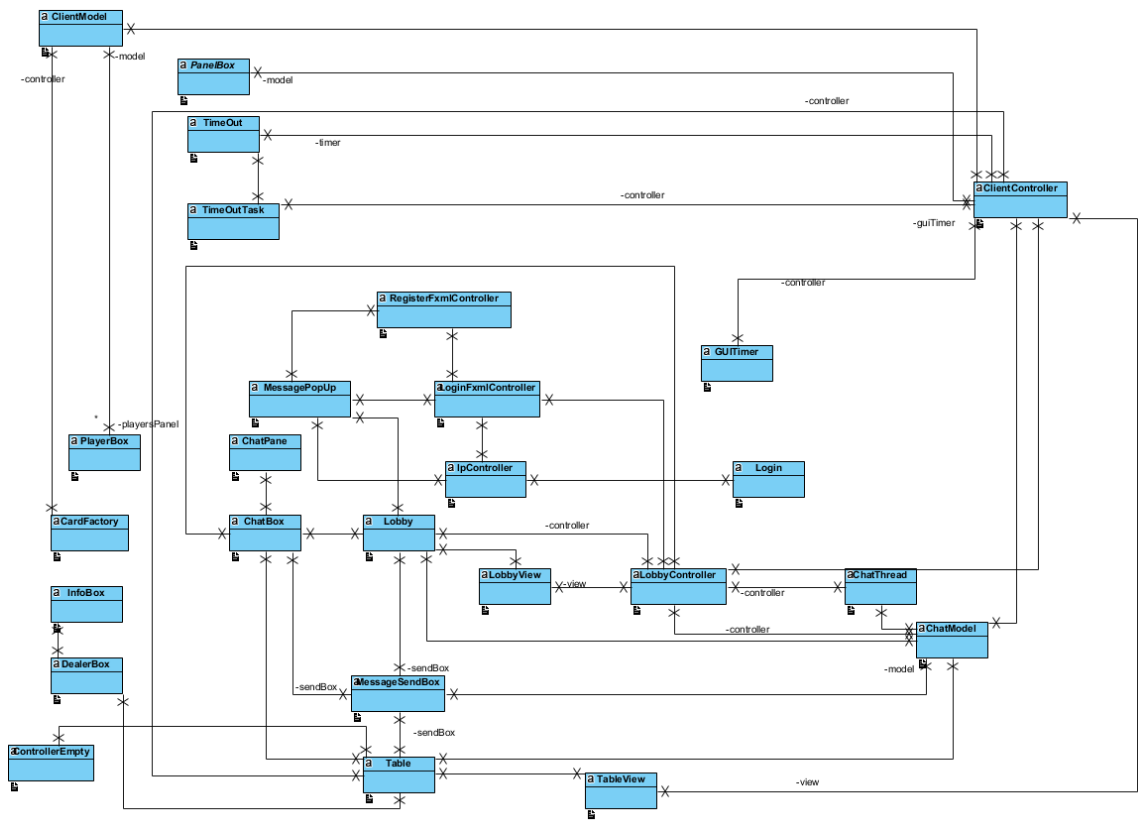
## Szerver oldal uml-diagramja

A digitális adathordozón nagyobb felbontásban megtalálhatók a diagramok.



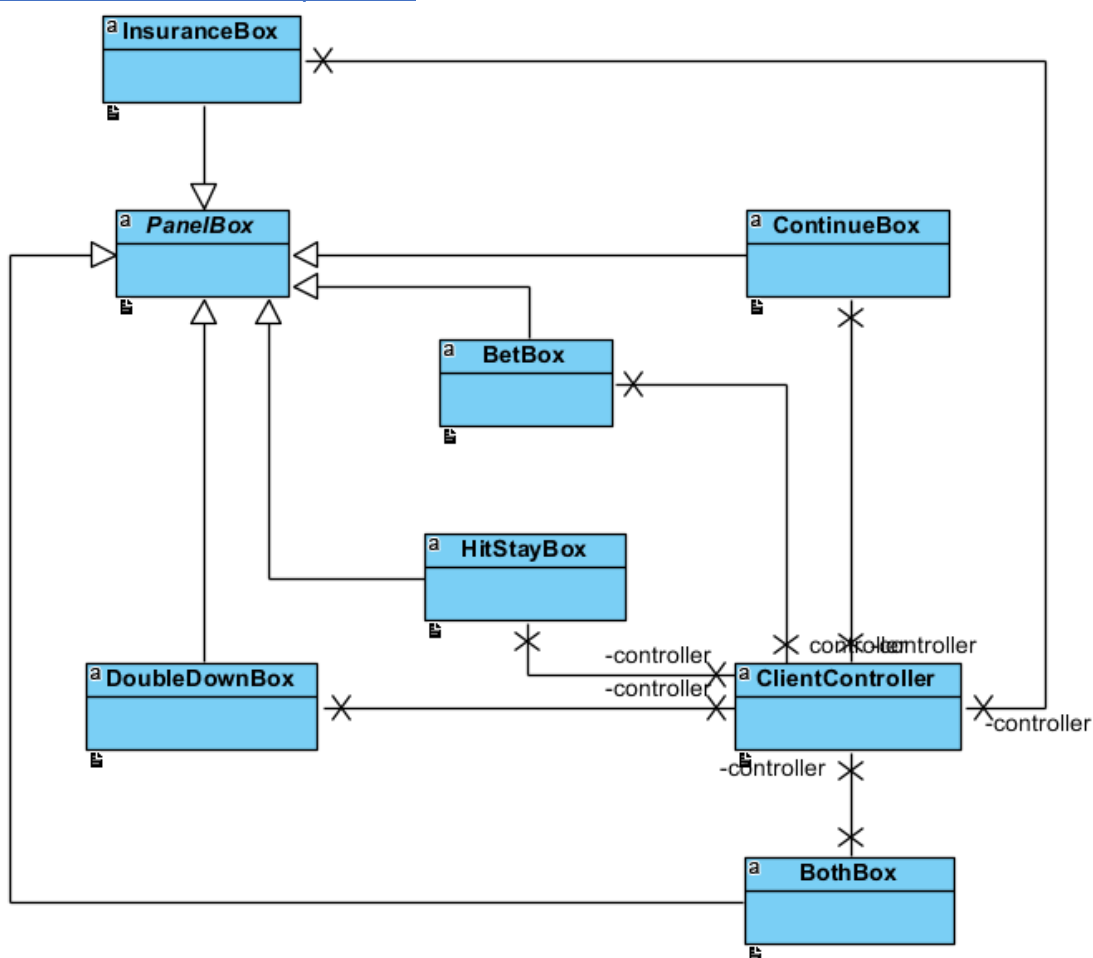
27. ábra

## Kliens oldal uml-diagramja



28. ábra

## PanelBox-ból leszármazó panelek



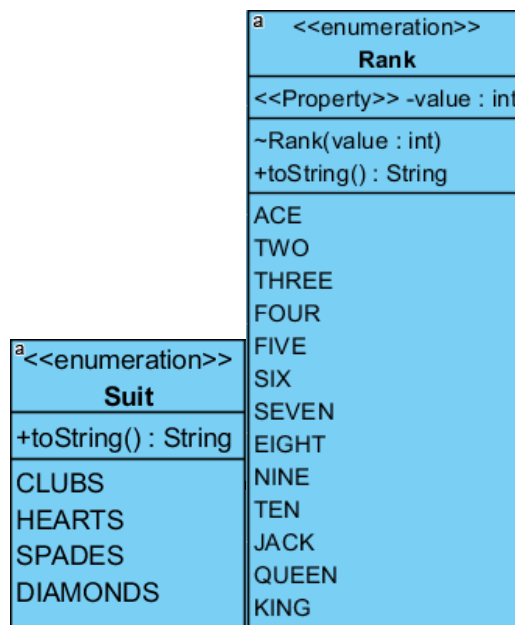
29. ábra

## A server.logic package

### Suit és Rank osztály

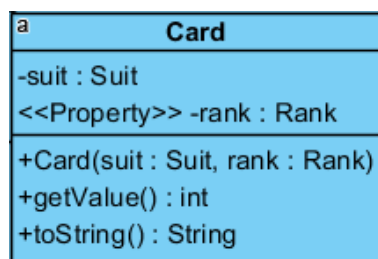
A lapok színének a megvalósítására létrehoztam egy Suit felsoroló típust, amelynek lehetséges értékei a CLUBS, HEARTS, SPADES, DIAMONDS.

Egy másik Rank felsoroló típust a lapok értékének hoztam létre aminek az értékei lehetnek ACE(1), TWO(2)...TEN(10),JACK(10)...KING(10) a zárójelben szereplő érték a lap értéke. Egy toString függvénnyel lekérhető a neve (kártyák előállítása miatt), illetve egy getValue függvénnyel elérhető a lap értéke.



30. ábra

## Card osztály



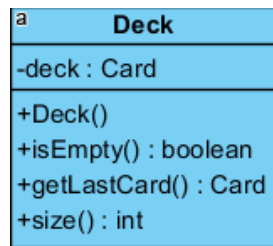
31. ábra

A Card osztály egy kártyát valósít meg.

### Fontosabb Adattagok:

- Suit suit: A kártya színét reprezentálja.
- Rank rank: A kártya értékét reprezentálja.

### Deck osztály



32. ábra

A Deck osztály legenerál mind a négy színből 13 különböző értékű kártyát (Card) és eltárolja azt.

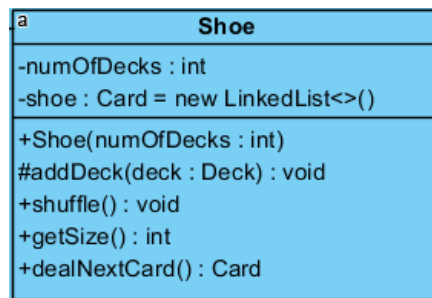
#### Fontosabb Adattagok:

- LinkedList<Card> deck: A kártyákat tároló lista.

#### Fontosabb Függvények:

- Card getLastCard(): Visszatér a pakli utolsó kártyájával és törli azt.

### Shoe osztály



33. ábra

A Blackjacknél kártya tároló reprezentálása ez egy olyan objektum, amiben bizonyos számú pakli van bekeverve.

#### Fontosabb Adattagok:

- LinkedList<Card> shoe: A bekevert paklikat tartalmazó lista.
- int numOfDecks: Hány pakliból álljon a Shoe.

#### Fontosabb Függvények:

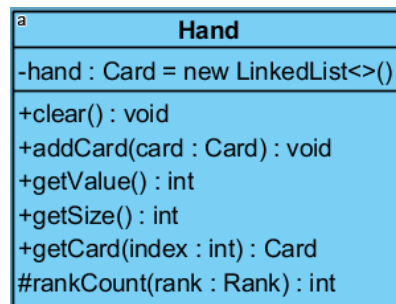
- Shoe(int numOfDecks): (konstruktor) Legenerál annyi paklit amennyi a paraméter.
- Card dealNextCard(): Visszatér a shoe utolsó lapjával és törli belőle azt.

#### Fontosabb Eljárások:

- shuffle(): Megkeveri a shoe-t.

- addDeck(Deck deck): Hozzáad egy paklit a shoe-hoz.

### Hand osztály



34. ábra

A játékos kezének az alaptulajdonságait reprezentálja.

#### Fontosabb Adattagok:

- LinkedList<Card> hand: A játékos kártyái.

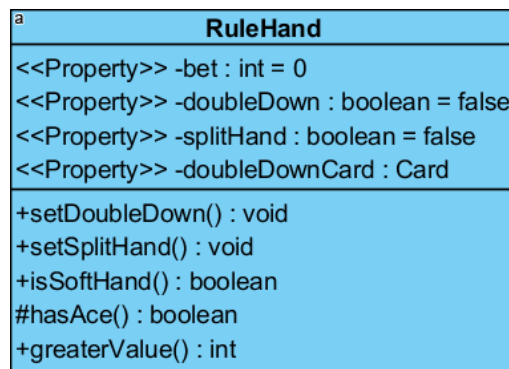
#### Fontosabb Függvények:

- int getValue() : A lapok értékével tér vissza.
- int getSize(): A lapok számával tér vissza.
- Card getCard(int index) : Az index helyen lévő kártyával tér vissza.
- int rankCount(Rank rank): Rank értékű kártyából hány darab van.

#### Fontosabb Eljárások:

- addCard(Card card): card kártya hozzáadása

### RuleHand osztály



35. ábra

A Hand osztályt terjeszti ki a játékhoz szükséges függvényekkel, metódusokkal adattagokkal.

#### Fontosabb Adattagok:

- int bet: A kézre vonatkozó tét
- boolean doubleDown: Duplázott-e a játékos
- Card doubleDownCard: A duplázásnál kapott kártya

#### Fontosabb Függvények:

- int greaterValue(): Ha van ász a játékos kezébe ász akkor annak az értéke 11 és nem 1 addig amíg nem lépi túl a 21-et, ez a függvény 11-gyel számolja az ász értékét és úgy számolja a lapok értékét.
- boolean isSoftHand(): Van-e ász a lapok között?

## Player osztály

```
a Player
-MAXIMUM_SCORE : int = 21
-BLACKJACK_PAYOUT : double = 1.5
-SERV_AUTH : String = "SERVERCOMMAND"
-DELIMITER : String = "-"
<<Property>> -in : BufferedReader
<<Property>> -out : PrintWriter
-money : double
-hasBlackjack : boolean = false
<<Property>> -username : String
-choice : String
-isAnswered : boolean = false
-insuranceBet : double = 0.0
-placedInsuranceBet : boolean = false
-startLatch : CountdownLatch
-betLatch : CountdownLatch
-insuranceBetLatch : CountdownLatch
-dealLatch : CountdownLatch
-dealerTurnLatch : CountdownLatch
-firstPlayerCardLatch : CountdownLatch
-secondPlayerCardLatch : CountdownLatch
-firstDealerCardLatch : CountdownLatch
-continuePlaying : boolean = true
<<Property>> -id : int
<<Property>> -socket : Socket
<<Property>> -lose : int
<<Property>> -win : int
<<Property>> -left : boolean = false
-timeOut : boolean
<<Property>> -table : ServerTable
-playerHands : RuleHand = new LinkedList<>()
<<Property>> -hand : RuleHand
<<Property>> -server : GameServer

+Player(id : int, username : String, money : double, win : int, lose : int)
+playBlackjack() : void
+reset() : void
+getBet() : void
+newRound() : void
-insuranceTurn() : void
+yourTurn(aHand : RuleHand) : void
-doubleDownOption(aHand : RuleHand) : void
-hitStayOption(aHand : RuleHand) : void
-doubleDown(aHand : RuleHand) : void
-hitStand(aHand : RuleHand) : void
-dealerTurn() : void
-resultTurn(aHand : RuleHand) : void
-continueTurn() : void
+getAnswer() : void
+send(rawString : String) : void
-numToString(num : double) : String
+startLatchCountDown() : void
+betLatchCountDown() : void
+insuranceBetLatchCountDown() : void
+dealLatchCountDown() : void
+firstPlayerCardLatchCountDown() : void
+dealerFirstCardLatchCountDown() : void
+secondPlayerCardCountDownLatch() : void
+dealerTurnLatchCountDown() : void
+hashCode() : int
+equals(obj : Object) : boolean
+getMoney() : Double
+run() : void
```

36. ábra

A játékost reprezentáló osztály, itt található egy játék kör és a játékosra vonatkozó logika. Ez az osztály határozza meg, hogy mi következik egy adott lépés után. Ez az osztály külön Thread-ként fut, a működése a játékos viselkedését szimulálja a kienstől vár választ és az alapján folytatja a játékot.



### Fontosabb Adattagok:

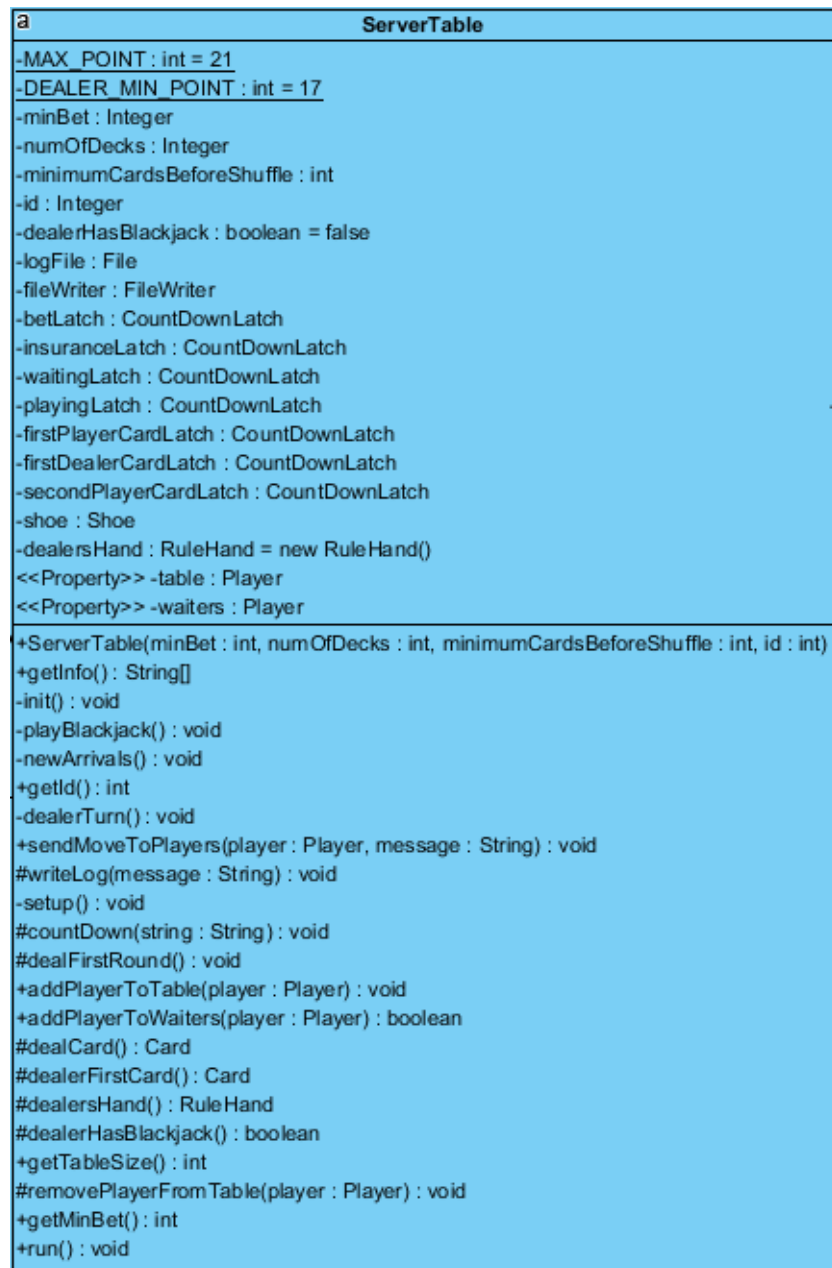
- static final double BLACKJACK\_PAYOUT: Blackjack esetén a kifizetési szorzó.
- static final String SERV\_AUTH: Szerver üzenetek prefixe.
- CountdownLatch startLatch: Az összes játékos készen áll-e (akkor 0).
- CountdownLatch betLatch: Az összes játékos megtette már a tétjét (akkor 0).
- CountdownLatch insuranceBetLatch: Kötöttek-e már biztosítást (akkor 0).
- CountdownLatch dealLatch: Kapott-e mindenki lapot (akkor 0)..
- CountdownLatch dealerTurnLatch: Osztóra várakozás
- CountdownLatch firstPlayerCardLatch: Játékosok első lapjára várakozó latch.
- CountdownLatch secondPlayerCardLatch: Játékosok második lapjára várakozó latch.
- CountdownLatch firstDealerCardLatch: Az osztó első lapjára várakozó latch
- boolean left: Kilépett-e a játékból.

### Fontosabb Eljárások:

- run(): Leírja mit csináljon az osztály a futása alatt, belépéskor küld egy üzenetet a kliensnek, ezután meghívja a playBlackjack() eljárást, ha a játékos nem folytatja a játékot küld egy üzenetet illetve, ha vissza akar lépni a Lobbyba akkor lekéri a szükséges adatokat. A játék elhagyása előtt tájékoztatja az asztalt.
- playBlackjack(): Egy kör pontos menetét írja le: törli az előző kör adatait; vár a többi játékosra, megteszi a tétet, vár a többi játékosra, vár a lapokra, vár a többi játékos lapjára, elkezdi a játékot, vár a többi játékosra, vár az osztóra, eredmények, körvége.
- reset(): Előző kör adatainak törlése.
- getBet(): A kliens oldalról várja a tétet és ellenőrzi, hogy nagyobb-e a minimum tétnél illetve van-e annyi pénzünk, ha igen beállítja megtett tétnek, ha nem újra kéri és tájékoztatja a klienst a hibáról.
- newRound(): Törli az előző játékból maradt lapokat és annak értékeit és erről tájékoztatja a klienst. Ezután ellenőrzi, hogy az osztó első lapja ász-e, ha igen megvárja a biztosításokat. Ha 21-e van az osztónak feltett tét kétszeres a nyeremény, a játék végén derül ki.

- `insuranceTurn()`: Ellenőrzi, hogy akar-e biztosítást kötni a játékos, ha igen akkor feltett tétjének a felét teszi meg (ha van elég pénze).
- `yourTurn(RuleHand aHand)` : Ha a játékos a soron következő választhat az első körnél, hogy dupláz-e, ha nem lapot kérhet addig ameddig nem veszít vagy megállhat a paraméter split hand-re vonatkozik, de még nincs implementálva funkció.
- `doubleDownOption(RuleHand aHand)`: A lapjai alapján választhat a játékos, hogy dupláz, lapot kér vagy megáll.
- `hitStayOption(RuleHand aHand)`: A lapjai alapján választhat a játékos lapot kér vagy megáll.
- `doubleDown(RuleHand aHand)`: Ha a játékos dupláz ez a függvény fut le a duplázás szabályai szerint.
- `hitStand(RuleHand aHand)`: A lapkérés és a megállásért felelős függvény
- `dealerTurn()`: Az osztó lapjainak elküldése illetve ha blackjackje van az osztónak akkor a biztosítási összeget is elküldi .
- `resultTurn(RuleHand aHand)` :A kör eredményeinek elküldése.
- `continueTurn()`: Ellenőrzi, hogy a játékos akar-e tovább játszani, ha nem akkor a játékos átkerül a Lobbyba.
- `getAnswer()`: A játékostól várja a válaszokat, ha kilép a játékos vagy valami hiba lép fel akkor LEAVE válasszal tér vissza és ez alapján a szerver kilépteti és lejátssza a játékot helyette.
- `send(String rawString)`: Rendezi és elküldi az üzenetet a kliensnek, illetve a logfájlba beleírja a játékos lépését.
- `*LatchCountDown()`: Egyet visszazámol.

## ServerTable osztály



37. ábra

Ez az osztály reprezentál egy asztalt, tartozik hozzá egy osztó, illetve egy játékkört valósít meg a körben minden folyamatot megismétel minden játékoson. Az asztal csak akkor végez műveleteket, ha legalább egy játékos játszik rajta. Egy asztalhoz négy játékos ülhet le. Minden asztal egy külön folyamat.

### Fontosabb Adattagok:

- final Integer minBet: Minimális tét az asztalon.
- numOfDecks: Paklik az asztalnál.
- final int minimumCardsBeforeShuffle: Shoe minimális mérete.
- final AdditionalHand dealersHand: Az osztó lapjai.
- final LinkedList<Player> table: Asztalnál lévő játékosok.
- final ArrayList<Player> waiters: A játékra váró felhasználók.
- File logFile: Logfile.
- FileWriter fileWriter: Logfile számára fenttartott filewriter.

### Fontosabb Metódusok:

- playBlackjack(): Megfeleltethető a Playerben található azonos nevű metódusnak, csak itt az asztal vár a játékosokra és nem a játékosok egymásra. Annyi kiegészítéssel, hogy itt a körvégén a játékosok adataival frissül az adatbázis, illetve új játékosok csatlakozhatnak (newArrivals metódus).
- setup(): Minden kör elején a változók alaphelyzetbe állítását végzi el.
- countDown (String string) string paraméter szerint a választott latch-en egyet visszaszámol.
- dealFirstRound() A játékosoknak és az osztónak oszt 2 lapot a metódus.
- addPlayerToTable(Player player): Hozzáad az asztalhoz egy új játékost és indít neki egy folyamatot.
- newArrivals(): Ez a metódus felelős a várakozó játékosok hozzáadásáért az asztalhoz.
- writeLog(String message): Ez a metódus a logfile-ba írja a játékos adatait.

### Fontosabb Függvények:

- Card dealCard(): Egy lappal tér vissza a shoe tetejéről, ha a shoe elfogyna, létrehoz egy új shoe-t.
- Card dealerFirstCard(): Az osztó első, ismert lapjával tér vissza.
- boolean addPlayerToWaiters(Player player): Ez a függvény hozzáadja a várakozókhoz a játékost, a visszatérési érték a hozzáadás sikerességét jelzi.

[A server.gameServer package](#)

## GameServer osztály

a	GameServer
	<pre>-SERV_AUTH : String = "SERVERLOBBY" -serverSocket : ServerSocket -serverPort : int -chatPort : int -serverListenerThread : Thread &lt;&lt;Property&gt;&gt; -onlinePlayers : Player &lt;&lt;Property&gt;&gt; -tables : ServerTable -chatServer : ChatServer -serverListener : ServerListener  +GameServer(serverPort : int, chatPort : int) -startServer() : void -createTables() : void +addTable(minBet : int, numOfDecks : int, minimumCardsBeforeShuffle : int) : void +addPlayersToTable(player : Player, index : int) : boolean +sendDataForLobby(player : Player) : void +addPlayer(socket : Socket, player : Player) : void -sendTableList(socket : Socket, out : PrintWriter) : void -sendLeaderBoard(socket : Socket, out : PrintWriter) : void -removeOfflineClient(socket : Socket) : void +getClientMessage(socket : Socket) : String +isOnline(username : String) : boolean +main(args : String[]) : void</pre>

38. ábra

Ez az osztály felel a játékszerver indításáért, ez az osztály hozza létre az asztalokat, illetve sikeres belépés után a felhasználónak ez az osztály biztosítja a játékos adatait és tárolja, hogy éppen ki játszik. Ennek az osztálynak a futtatásával indul el a szerver.

### Fontosabb Adattagok:

- ServerSocket serverSocket: Szerver socket
- final int serverPort: Játékszerver portja
- final int chatPort: Chatszerver portja
- List<Player> onlinePlayers: Online játékosok
- List<ServerTable> tables: Asztalok
- ChatServer chatServer: Chatszerver
- ServerListener serverListener: Játékosokat fogadó osztály
- Thread serverListenerThread Játékosokat fogadó folyamat

GameServer(int serverPort,int chatPort): Beállít két portot egyet a chat-,egyét a játékszervernek.

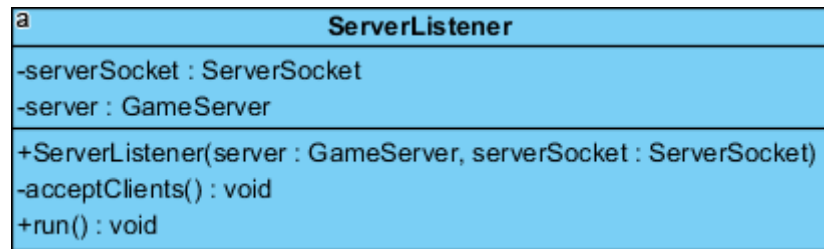
#### Fontosabb Eljárások:

- startServer(): Ez az eljárás felel a szerver indításáért indít egy új folyamatot a kliensek fogadására,
- createTable():Létrehozza a játék asztalokat.
- sendDataForLobby(Player player): A kliensnek elküldi a Lobbyhoz tartozó információkat.
- addPlayer(Socket socket,Player player): Ez az eljárás validáció után hozzáadja a játékos(player) az online játékosokhoz és elküldi a kezdés előtti információkat.
- sendTableList(Socket socket,PrintWriter out):Ez az eljárás elküldi az asztalok listáját.
- sendLeaderBoard(Socket socket,PrintWriter out): Ez az eljárás elküldi a leaderboard-ot.
- removeOfflineClient(Socket socket): A paraméterben kapott socket-tel rendelkező játékos törli az online játékosok közül.
- main(String[] args): A futtatásért felelős main metódus.

#### Fontosabb Metódusok:

- String getClientMessage(Socket socket):A kliens üzenetét fogadó metódus esetleges hibák kezelése.
- boolean isOnline(String username): A username felhasználónevű játékos online van-e?

### ServerListener osztály



39. ábra

Ez a folyamat felel a kliens kapcsolódásáért. Hallgat a szerver-porton és ha egy kliens csatlakozik létrehoz neki egy új folyamatot, ahol a validáció történik.

#### Fontosabb Adattagok:

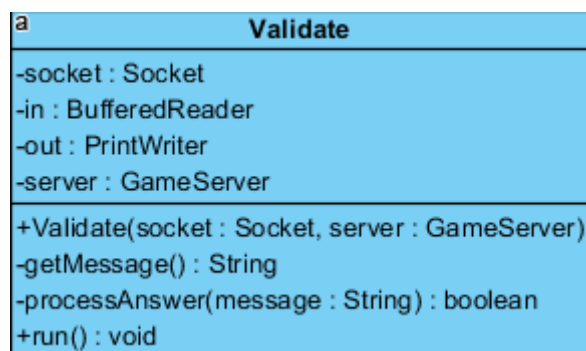
- final GameServer server: Játékszerver
- final ServerSocket serverSocket: Szerver socket

A konstruktor a paramétereket elmenti.

#### Fontosabb Eljárások:

- acceptClients():Hallgat a megadott porton, ha új kliens érkezik új validációs folyamatot indít a Validate osztállyal.

### Validate osztály



40. ábra

Ez az osztály felel azért, hogy ellenőrizze a játékos bejelentkezési adatai vagy a regisztrációs adatai megfelelőek, ha nem akkor tájékoztatja a hibás adatról a klienst, ha megfelelő átadja az adatait a WaitingRoom osztálynak.

#### Fontosabb Adattagok:

- GameServer server: a játékszerver

A konstruktor elmenti a paramétereket és megpróbálja a socket ki- és bemenetét elérni.

#### Fontosabb Függvények:

- String getMessage(): A kliens válaszára váró függvény.
- boolean processAnswer(String message): A message üzenet alapján megpróbálja validálni az adatokat, ha bejelentkezés történik ellenőrzi, hogy a jelszó egyezik-e a tárolt hash-sel és létezik-e a felhasználó illetve már nincs-e online. Regisztrációnál ellenőrzi, hogy a szükséges mezők egyediek-e (email, username), ha az adatok megfelelőek akkor, létre hozz egy új sort az adatbázisban. Ha sikeres a belépés akkor az adatbázis alapján legenerál egy új játékost és átadja a WaitingRoom osztálynak. Hibaesetén értesíti a klienst.

#### WaitingRoom osztály

Ez az osztály a kientől várja a választ arra vonatkozóan melyik asztalhoz szeretne csatlakozni, majd a szerveren keresztül megpróbál csatlakozni az asztalhoz, ha nem sikerül, akkor az osztály tovább vár egy megfelelő válaszra.

#### Fontosabb Adattagok:

- GameServer server: A játékszerver
- Player player: A játékost szimbolizáló változó

A konstruktor elmenti, illetve beállítja az adattagokat.

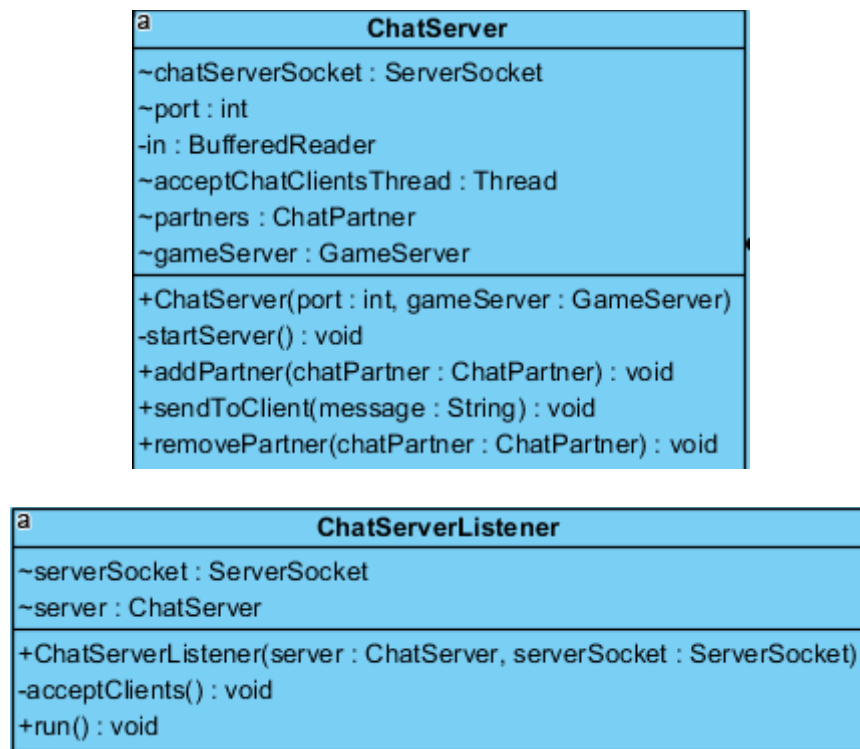
#### Fontosabb Eljárások:

- getChoice() : Feldolgozza a kliens választ addig fut ameddig nem választ egy olyan asztalt, amihez tud csatlakozni.



## [A server.chatServer](#)

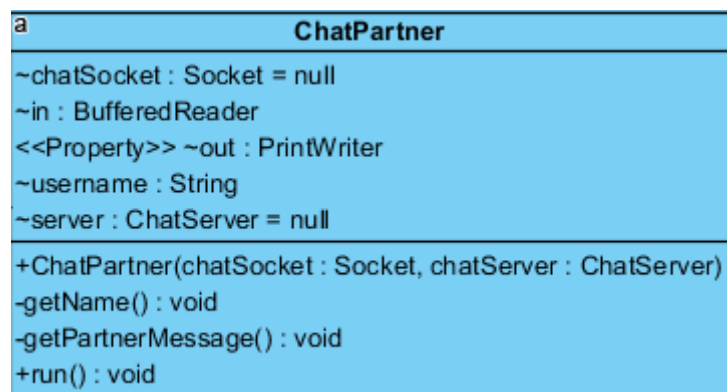
### ChatServer és ChatServerListener osztály



41. ábra

Ez a két osztály majdnem teljesen megegyezik a `gameServer` osztályban található két játék osztállyal csak funkcióját tekintve a chat-ért felelős.

### ChatPartner osztály



42. ábra

Egy kliens üzenet feldolgozásáért felelős osztály. Először beállítja a kliens nevét utána a tőle jövő üzeneteket továbbítja a játékosnak. Az osztály addig fut ameddig a socket kapcsolata aktív.

## A server package

### BCrypt és BcryptHashing osztály

Nem saját osztályok. A jelszó titkosításáért, illetve a titkosított jelszó összehasonlításáért felelős osztályok. Copyright (c) 2006 Damien Miller [djm@mindrot.org](mailto:djm@mindrot.org).<sup>[7]</sup>

### DatabaseInitializer osztály

a	DatabaseInitializer
<pre>-connection : Connection -properties : Properties -userdata : List&lt;List&lt;String&gt;&gt; = new ArrayList&lt;&gt;() -DATABASE_URL : String = "jdbc:mysql://localhost:3306/blackjack?zeroDateTimeBehavior=convertToNull&amp;serverTimezone=UTC" -databaseUrl : String -DELETE_PLAYER_TABLE_SQL : String = "DROP TABLE PLAYER" -DELETE_USERDATA_TABLE : String = "DROP TABLE USERDATA" -CREATE_PLAYER_TABLE_SQL : String = "CREATE TABLE PLAYER ("     + "ID INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY ,"     + "USERNAME VARCHAR(100) NOT NULL,"     + "MONEY DOUBLE(10,3),"     + "WIN SMALLINT,"     + "LOST SMALLINT"     + ")" -CREATE_USERDATA_TABLE_SQL : String = "CREATE TABLE USERDATA("     + "ID INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY ,"     + "USERNAME VARCHAR(100),"     + "PASSWORD CHAR(60) NOT NULL,"     + "EMAIL VARCHAR(255) NOT NULL,"     + "NAME VARCHAR(255) NOT NULL,"     + "BIRTH DATE NOT NULL,"     + "REGISTER DATE NOT NULL"     + ")"  -DatabaseInitializer() -DatabaseInitializer(host : String, port : int) +init() : void -deleteTables() : void -createTables() : void -uploadUserTable() : void -close(statement : Statement) : void -createUserData(username : String, password : String, email : String, name : String, birhDate : String) : ArrayList +main(args : String[]) : void</pre>	

43. ábra

Ez az osztály felel az adattáblák létrehozásáért, a localhost 3306 portján működik. Létrehozza a szükséges táblákat (törli a régieket) illetve feltölti teszt játékosokkal. Egy

JDBC connectorral csatlakozik az adatbázishoz és a teszt adatokat a DAO-n keresztül hozza létre.

## [A dao package](#)

### DefaultDao<T extends Player> osztály

```
a                                     DefaultDao
#DATABASE_URL : String = "jdbc:mysql://localhost:3306/blackjack?zeroDateTimeBehavior=convertToNull&serverTimezone=UTC"
#connection : Connection
#properties : Properties

+DefaultDao()
+findAllPlayers() : List<T>
+findPlayerByUsername(username : String) : T
+findPlayerById(id : Integer) : T
+alreadyContains(username : String, email : String) : String
+findLoginDatas(username : String) : List<String>
+createPlayer(username : String, password : String, email : String, name : String, birthDate : String) : void
+update(entity : T) : void
+delete(id : int) : void
#fromDataasDataTable(username : String, password : String, email : String, name : String, birthDay : String) : Statement
#fromDataasPlayerTable(resultSet : ResultSet) : Statement
#getInsertDataSql() : String
#getFindDataByUsernameSql() : String
#getUpdatePasswordSql() : String
#getFindPlayerByUsername() : String
#getFindAllPlayerSql() : String
#getFindPlayerByIdSql() : String
#getInsertPlayerSql() : String
#getUpdatePlayerSql() : String
#getDeletePlayerSql() : String
#getUpdateMoneySql() : String
#getUpdateWinLoseSql() : String
#getFindIdByUsername() : String
#getFindPasswordByUsername() : String
#getFindDataByEmail() : String
#fromResultSet(resultSet : ResultSet) : T
#fromEntity(query : String, entity : T, withId : boolean) : Statement
#getDeleteUserDataSql() : String
#close(statement : Statement) : void
```

44. ábra

A Player játékoshoz tartozó DAO objektum, ami az adatbázis műveleteket bonyolítja le. Egy JDBC connector-ral csatlakozik a mySQL adatbázishoz.

#### Fontosabb adattagok:

- static final String DATABASE\_URL: az adatbázis URL-je (localhost 3306 portján működik).

#### Fontosabb Függvények:

- T findPlayerByUsername(String username): Megkeresi az adott felhasználónevű játékost és visszatér vele.
- T findPlayerById(Integer id): Megkeresi az adott azonosítójú játékost.
- String alreadyContains(String username,String email): Megvizsgálja, hogy szerepel-e az adatbázisban az email vagy a jelszó és egy hibaüzenettel tér vissza ha igen ,ha nem egy konfirmációsüzenettel .
- List<String>findLoginDatas(String username):Megkeresi az adott felhasználó névhez tartozó adatokat.

#### Fontosabb Metódusok:

- delete(int id): Törli a megadott azonosítóval rendelkező felhasználót az adatbázisból.
- update(T entity): Az adott felhasználó adatait frissíti az adatbázisban a paraméterben kapott objektum alapján.
- createPlayer(String username,String password,String email,String name,String birthDate): Létrehoz egy játékost az adatbázisban.

## PlayerDao extends DefaultDao<Player> osztály

a	<b>PlayerDao</b>
-INSERT_DATA_SQL : String = "INSERT INTO USERDATA (USERNAME, PASSWORD, EMAIL, NAME, BIRTH, REGISTER) VALUES(?,?,?, ?, ?, ?)"	
-FIND_DATA_BY_USERNAME_SQL : String = "SELECT * FROM USERDATA WHERE LIKE = ?"	
-UPDATE_PASSWORD_SQL : String = "UPDATE USERDATA SET PASSWORD = ? WHERE ID = ?"	
-FIND_PLAYER_BY_USERNAME_SQL : String = "SELECT * FROM PLAYER WHERE USERNAME = ?"	
-FIND_ALL_PLAYER_SQL : String = "SELECT * FROM PLAYER"	
-FIND_PLAYER_BY_ID_SQL : String = "SELECT * FROM PLAYER WHERE ID = ?"	
-INSERT_PLAYER_SQL : String = "INSERT INTO PLAYER(USERNAME, MONEY, WIN, LOST) VALUES(?,?, ?, ?)"	
-UPDATE_PLAYER_SQL : String = "UPDATE PLAYER SET MONEY = ?, WIN = ?, LOST = ? WHERE ID = ?"	
-DELETE_PLAYER_SQL : String = "DELETE FROM PLAYER WHERE ID = ?"	
-DELETE_USERDATA_SQL : String = "DELETE FROM USERDATA WHERE ID = ?"	
-UPDATE_MONEY_SQL : String = "UPDATE PLAYER SET MONEY = ? WHERE ID = ?"	
-UPDATE_WIN_LOSE_SQL : String = "UPDATE PLAYER SET WIN = ?, LOST = ? WHERE ID = ?"	
-FIND_ID_BY_USERNAME_SQL : String = "SELECT ID, USERNAME FROM USERDATA WHERE USERNAME LIKE ?"	
-FIND_PASSWORD_BY_USERNAME : String = "SELECT USERNAME, PASSWORD FROM USERDATA WHERE USERNAME LIKE ?"	
-FIND_DATA_BY_EMAIL : String = "SELECT ID FROM USERDATA WHERE EMAIL = ?"	
<<Property>> -startingMoney : double = 2500.0	
-PlayerDao() +getInstance() : PlayerDao #fromResultSet(resultSet : ResultSet) : Player #fromEntity(query : String, player : Player, withId : boolean) : Statement #fromDatasDataTable(username : String, password : String, email : String, name : String, birthDay : String) : Statement #fromDatasPlayerTable(resultSet : ResultSet) : Statement #getInsertDataSql() : String #getFindDataByUsernameSql() : String #getUpdatePasswordSql() : String #getFindPlayerByUsername() : String #getFindAllPlayerSql() : String #getFindPlayerByIdSql() : String #getInsertPlayerSql() : String #getUpdatePlayerSql() : String #getDeletePlayerSql() : String #getUpdateMoneySql() : String #getUpdateWinLoseSql() : String #getFindIdByUsername() : String #getDeleteUserDataSql() : String #getFindPasswordByUsername() : String #getFindDataByEmail() : String	

45. ábra

A DefaultDao osztályt terjeszti ki. Itt tároljuk az adatbázis kapcsoltakhoz szükséges SQL utasításokat. A statikus adattagok az SQL utasítások. Ezen kívül egy adattag van a játékosok regisztráláskor kapott pénze az int startingMoney. Tartalmaz egy statikus DefaultPlayerDaoHolder osztályt melynek egy INSTANCE adattagja van, ami egy PlayerDao példánnyal tér vissza;

### Fontosabb Függvények:

- static PlayerDao getInstance() : Az osztály példányosításával tér vissza.
- Player fromResultSet(ResultSet resultSet): A resultSet halmazból létrehoz egy játékost .
- Statement fromEntity(String query, Player player, boolean withId): Egy játékos adataiból létrehoz egy Statement-et egy adatbázisművelethez.

- Statement fromDatasDataTable(String username,String password,String email,String name,String birthDay): A paraméterekből létrehoz egy Statement-et egy adatbázisművelethez .
- Statement fromDatasPlayerTable(ResultSet resultSet): A resultSet halmazból létrehoz egy Statement-et a player táblából egy adatbázisművelethez .

## [A view.panels](#)

### Absztrakt PanelBox extends VBox osztály

a	<b>PanelBox</b>
	+PanelBox() #setLabelFont(label : Label) : void #createButton(name : String) : Button #makeNumeralTextField(prompt : String) : TextField #setSize() : void #init() : void #setEventHandlers() : void

46. ábra

Ez az osztály felel a GUI irányításáért felelős panelekért. Ebből az osztályból származnak le az irányító panelek.

- BetBox osztály: felelős a téttrakás folyamataért
- BothBox osztály: A hit, stand, double down vagy split hand opciók választásáért
- ContinueBox osztály: A játék végén a folytatásért
- DoubleDownBox osztály: A hit, stand, double down opciók választásáért
- HitStayBox osztály: A hit, stand opciók választásáért
- InsuranceBox osztály: Biztosításért felelős

a	<b>BetBox</b>	a	<b>BothBox</b>
	- betBoxList : ObservableList - betList : ObservableList - betLabel : Label - betButton : Button - betBox : HBox - betTextField : TextField ~ controller : ClientController + BetBox(controller : ClientController) + init() : void # setEventHandlers() : void		- bothList : ObservableList - bothLabel : Label - splitButton : Button - hitButton : Button - stayButton : Button - doubleDownButton : Button - bothHolder : HBox - bothHolderList : ObservableList - controller : ClientController + BothBox(controller : ClientController) + init() : void # setEventHandlers() : void

<b>a ContinueBox</b>	<b>a DoubleDownBox</b>
-countinueList : ObservableList -continueLabel : Label -noButton : Button -yesButton : Button -continueHolder : HBox -continueHolderList : ObservableList -controller : ClientController	-doubleDownList : ObservableList -bothLabel : Label -hitButton : Button -stayButton : Button -doubleDownButton : Button -doubleDownHolder : HBox -doubleDownHolderList : ObservableList -controller : ClientController
+ContinueBox(controller : ClientController) +init() : void #setEventHandlers() : void	+DoubleDownBox(controller : ClientController) +init() : void #setEventHandlers() : void

<b>a HitStayBox</b>	<b>a InsuranceBox</b>
-hitStayList : ObservableList -hitLabel : Label -hitButton : Button -stayButton : Button -hitStayHolder : HBox -hitStayHolderList : ObservableList -controller : ClientController	-insuranceList : ObservableList -insuranceLabel : Label -insuranceHolder : HBox -noButton : Button -yesButton : Button -insuranceHolderList : ObservableList -controller : ClientController
+HitStayBox(controller : ClientController) +init() : void #setEventHandlers() : void	+InsuranceBox(controller : ClientController) +init() : void #setEventHandlers() : void

47. ábra

További panelek:

- ChatBox osztály: A chat megjelenítésére szolgáló panel
- DealerBox osztály: Az osztó lapjait tartalmazó, illetve az információs panelt tartalmazó panel.
- ChatPane osztály: A ChatBox része az üzeneteket megjelenítő panel.
- MessageSendBox osztály: A ChatBox része ebből a panelből küldhetünk üzenetet.
- InfoBox osztály: Az információk kiírásáért felelős osztály.
- ControllerEmpty osztály: Egy szegéllyel rendelkező AnchorPane egy üres panel, ami akkor jelenik meg a játékosnak, ha nem ő a soron következő.

a	DealerBox
	-dealerLabel : Label -handValueLabel : Label -dealerHand : HBox -holder : HBox -holderList : ObservableList ~dealerHandList : ObservableList ~list : ObservableList -infoBox : InfoBox
	+DealerBox() +init() : void -setVbox(vbox : VBox) : void +addCard(card : Label) : void +removeFaceDownCard() : void +newRound() : void +setHandValue(value : String) : void +setInfo(info : String) : void +setTime(time : String) : void

a	InfoBox	a	MessageSendBox
	-infoLabel : Label -logLabel : Label -infoList : ObservableList -timeLabel : Label		-messageTextField : TextField -sendButton : Button -sendList : ObservableList -model : ChatModel
	+InfoBox() +init() : void +setInfo(info : String) : void +setTime(time : String) : void -setLabelFont(label : Label) : void		+MessageSendBox(model : ChatModel) -init() : void -createButton(prompt : String) : Button -setActionListener() : void

a	ChatPane	ControllerEmpty
	~label : Label	
	+ChatPane(label : Label)	+ControllerEmpty()

a	ChatBox
	-chatList : ObservableList <<Property>> -chatPane : ChatPane -sendBox : MessageSendBox
	+ChatBox(chatPane : ChatPane, sendBox : MessageSendBox) -init() : void



## Table osztály és a TableView osztály

a	Table
	-id : int -playersBox : HBox -players : ObservableList -controllBox : AnchorPane <<Property>> -controller : ClientController -chatModel : ChatModel -sendBox : MessageSendBox -betBox : BetBox -bothBox : BothBox -hitStayBox : HitStayBox -insuranceBox : InsuranceBox <<Property>> -dealerBox : DealerBox -doubleDownBox : DoubleDownBox -continueBox : ContinueBox -chatBox : ChatBox
	+Table(id : int, chatModel : ChatModel, controller : ClientController) +init() : void +addNewPlayer(playerBox : PlayerBox) : void +addChatBox(chatBox : ChatBox) : void +setEmpty() : void +setBet() : void +setInsurance() : void +setDoubleDown() : void +setHitStay() : void +setContinue() : void +setTime(time : String) : void -findPlayerById(id : int) : PlayerBox +changePlayerBox(id : String, playerBox : PlayerBox) : void +findIndex(id : int) : int +removePlayer(id : String) : void +addCardToDealerHand(url : String) : void +removeDealerHand() : void +addCardToPlayerFirstHand(id : int, url : String) : void +newRound() : void +setMoneyForPlayer(id : int, money : Double) : void -setUpPanels() : void +setDealerHandValue(value : String) : void +setInfo(info : String) : void +exit() : boolean

a	TableView
	+YELLOW_FONT_COLOR : Color = new Color(0.91, 0.60, 0.11, 1) +GAME_FONT_TYPE : Font = Font.font("Times New Roman", FontWeight.BOLD, 14) +CSS_LAYOUT : String = "-fx-padding: 10;" + "-fx-border-style: solid inside;" + "-fx-border-width: 2;" + "-fx-border-insets: 5;" + "-fx-border-radius: 5;" + "-fx-border-color: black;" + "-fx-background-color: rgb(0, 91, 19)"
	+SCREEN_RES : Rectangle2D = Screen.getPrimary().getVisualBounds() +SHADOW : DropShadow = new DropShadow()
	~scene : Scene -table : Table
	+TableView(table : Table) +start(primaryStage : Stage) : void

A Table osztály felelős az GUI megjelenéséért ez az osztály írja le az asztalt és a modell-en keresztül küldi el az utasításokat a szervernek és a TableView osztály jeleníti meg.

Fontosabb Adattagok:

- final ClientController controller: Az irányítóosztály
- final ChatModel chatModel: Chat modellje
- final int id: A játékos és egyben a panel azonosítója
- HBox playersBox: A játékosokat megjelenítő box
- ObservableList players: Az előző adattag módosítólistája
- VBox controllBox egy üres boksx ennek a helyére kerülnek az irányítópanelek

A következő adattagok a fentebb említett panelek példányai:

- MessageSendBox sendBox;
- BetBox betBox
- BothBox bothBox
- HitStayBox hitStayBox
- InsuranceBox insuranceBox
- DealerBox dealerBox
- DoubleDownBox doubleDownBox
- ContinueBox continueBox

Fontosabb Eljárások:

- init(): Tábla előszítése.
- addNewPlayer(PlayerBox playerBox): Új játékost ad a táblához.
- addChatBox(ChatBox chatBox): ChatBox-ot ad a táblához

A következő eljárások a nevében szereplő funkciókat végrehajtó paneleket cseréli a tábla bal oldalán:

- setBet()
- setInsurance()
- setDoubleDown()
- setBoth()

- setHitStay()
- setContinue()
- changePlayerBox(String id, PlayerBox playerBox): Lecseréli az id azonosítójú playerBoxot a paraméterben megadottra.
- removePlayer(String id): Törli az adott azonosítójú játékos paneljét.
- addCardToDealerHand(String url): Egy kártyát generál és az osztó kezébe adja
- setMoneyForPlayer(int id, Double money): Beállítja az adott azonosítójú játékos pénzét.
- addCardToPlayerFirstHand(int id, String url): Egy kártyát generál és az adott azonosítójú játékos kezébe adja.
- newRound(): Felkészíti a táblát egy új körre.
- setUpPanels(): A játék során használt panelek létrehozása.
- setDealerHandValue(String value): Az osztó lapjainak értékének beállítása
- setInfo(String info): Információs panel beállítása.

Fontosabb Függvények:

- PlayerBox findPlayerById(int id): Az adott indexű játékos paneljával tér vissza
- int findIndex(int id): Megkeresi az adott azonosítójú játékos players lista indexét.
- boolean exit(): Bezárja a GUI-t.

## Lobby és a LobbyView osztály

a	Lobby
	<pre>-leaderBoardPane : ScrollPane -money : String -username : String -holder : HBox -tablePickerHolder : HBox -informationBox : VBox -informationList : ObservableList -holderList : ObservableList #SCREEN_RES : Rectangle2D = Screen.getPrimary().getVisualBounds() -tablePickerBox : ComboBox -pickerButton : Button -tablePickerLabel : Label -tablePickerList : ObservableList -tablePickerHolderList : ObservableList -leaderBoardBox : VBox -win : String -lose : String -leaderBoardList : ObservableList -numButton : Button -usernameButton : Button -moneyButton : Button -winButton : Button -loseButton : Button -leaderBoard : List&lt;List&lt;String&gt;&gt; -lineBoard : HBox -lineList : ObservableList &lt;&lt;Property&gt;&gt; -chatBox : ChatBox &lt;&lt;Property&gt;&gt; -chatModel : ChatModel -sendBox : MessageSendBox &lt;&lt;Property&gt;&gt; -controller : LobbyController  +Lobby(username : String, money : String, win : String, lose : String, chatModel : ChatModel, controller : LobbyController) +init() : void -createButton(name : String) : Button -setInformation() : void -setInformationLabel(str : String) : void -setText(label : Label) : void +updateInfo(money : String, win : String, lose : String) : void +setComboBox(tables : List&lt;List&lt;String&gt;&gt;) : void -setUpActionListener() : void +setUpLeaderBoard(list : List&lt;List&lt;String&gt;&gt;) : void -setLabelBorder(label : Label) : void -setButtonBorder(button : Button) : void +setChatPanel(chatPane : ChatPane) : void -waiting(owner : Window, pickedTable : String) : void</pre>

a	LobbyView
	<pre>-scene : Scene -lobby : Lobby  +LobbyView(lobby : Lobby) +start(primaryStage : Stage) : void</pre>

50. ábra

Ez a két osztály felel a Lobby kinézetért és megjelenéséért, érdemi számítás nem történik csak megjelenítés, illetve adattárolás.

## PlayerBox osztály



51. ábra

Ez az osztály jelenít meg egy játékost az asztalon.

### Fontosabb Adattagok:

- final String username: Felhasználónév
- final Integer id: Azonosító
- final String money: A játékos pénze.

Megjelenítő címkék:

- Label minimumLabel
- Label usernameLabel
- Label handValueLabel
- Label betLabel
- Label winLabel

- Label moneyLabel

Játékosok kezeit megjelenítő Boxok:

- HBox firstCardHolder
- HBox secondCardHolder
- VBox infoBox : Információs panel
- VBox splitHand: Nincs implementálva a split hand

Módosító listák:

- ObservableList playerBoxList
- ObservableList firstCardList
- ObservableList secondCardList
- ObservableList infoList
- ObservableList splitHandList
- int firstHandValue,secondHandValue: Adott kéz kártyáinak értéke
- StringBuilder winMessage: Győzelem esetén a szöveg
- primaryScreenBounds: A megjelenítő monitor méretei

A konstruktor PlayerBox(int id,String username,String money) elmenti a játékos adatait ezután előkészíti a panelt az init()-tel.

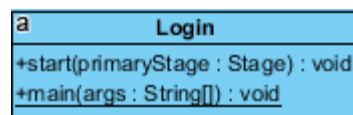
#### Fontosabb Eljárások:

- setHandValue(String index,String value): Beállítja az adott indexű kéznek az értékét.
- setBetValue(String bet): Beállítja az adott licitet a kézre.
- addCardToFirstHand(Label card): Hozzáad egy lapot a játékos kezéhez.
- addHand(String index): Hozzá ad a játékoshoz még egy kezet(paklit) az adott indexre.
- removeHand(String index): Eltávolítja a megadott kezet.
- removeCardFromHand(String handIndex): Eltávolítja az utolsó lapot a megadott kézből.
- addCardToHand(String index,String url): Hozzáad egy lapot a megadott kézhez.
- addCardToSecondHand(Label card): Hozzáad egy lapot a játékos 2.kezéhez
- newRound(): Törli a játékos kezeit.

## A login package

Az itt található osztályok és fxml fájlok a bejelentkezési ablakokat tartalmazzák, csak adatokat kérnek be, illetve, ha szervertől hibaüzenetet kapnak megjelenítik azt a felhasználó számára.

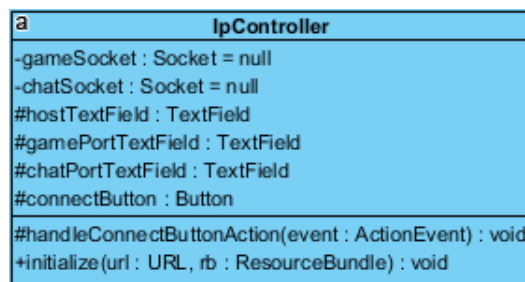
### Login osztály



52. ábra

Ennek az osztálynak a main metódusának meghívásával indul el a program. A start metódus jeleníti meg szerver adatainak megadására szolgáló ablakot.

### IpController osztály



53. ábra

A szerver adatainak megadására szolgáló ablak controller osztálya. Ha sikeres a csatlakozás a szerverhez, akkor a megjeleníti a bejelentkezési felületet.

### LoginFXMLController osztály

a	LoginFXMLController
	<pre>-gameSocket : Socket = null -chatSocket : Socket = null -out : PrintWriter -chatOut : PrintWriter -in : BufferedReader -usernameField : TextField -loginPasswordField : PasswordField -signButton : Button -registerButton : Button -lobbyController : LobbyController  #handleSubmitButtonAction(event : ActionEvent) : void #handleRegisterButtonAction(event : ActionEvent) : void +initialize(location : URL, resources : ResourceBundle) : void +setSockets(gameSocket : Socket, chatSocket : Socket) : void -getServerMessage() : String +leaveGame() : void</pre>

54. ábra

A bejelentkezési felület controller osztálya. Ha sikeres a bejelentkezés, ez az osztály indítja el a LobbyController-t. Ha a regisztráció gombra kattintunk megjeleníti a regisztrációs felületet.

### RegisterFXMLController osztály

a	RegisterFXMLController
	<pre>~value : LocalDate -gameSocket : Socket = null -chatSocket : Socket = null -out : PrintWriter -in : BufferedReader -nameTextfield : TextField -usernameTextfield : TextField -emailAgainTextfield : TextField -emailTextfield : TextField -passwordField : PasswordField -passwordAgainField : PasswordField -signUpButton : Button -backButton : Button -birthdayField : DatePicker  #handleSignUpButtonAction(event : ActionEvent) : void #handleBackButtonAction(event : ActionEvent) : void +initialize(url : URL, rb : ResourceBundle) : void +setSockets(gameSocket : Socket, chatSocket : Socket) : void -getServerMessage() : String +leaveGame() : void</pre>

55. ábra

A regisztrációs felület controller osztálya. Ha a játékos a visszalépés gombra kattint vagy sikeres a regisztráció a bejelentkezési felületet nyitja meg.



## MessagePopUp osztály

a	<b>MessagePopUp</b>
+popMessage(type : AlertType, owner : Window, title : String, prompt : String) : void	

56. ábra

Ez az osztály felelős a felugró ablakok előállításáért.

## A client package

### TimeOut és TimeOutTask osztály

a	<b>TimeOut</b>
-timer : Timer	
+TimeOut(controller : ClientController, command : String)	
+close() : void	

a	<b>TimeOutTask</b>
-command : String	
-controller : ClientController	
-timeOut : TimeOut	
#TimeOutTask(controller : ClientController, timeOut : TimeOut, command : String)	
+run() : void	

57. ábra

A TimeOut osztály meghívásakor létrehoz egy időzítőt, ami 15 másodperc elteltével lép a játékos helyett a paraméterben kapott parancsot átadja a TimeOutTask-nak ami elküldi a szervernek, leállítja az időzítőt és az irányító panelt kiüríti.

### GUITimer osztály

a	<b>GUITimer</b>
-running : AtomicBoolean	
-controller : ClientController	
+GUITimer(controller : ClientController)	
+setRunning(running : boolean) : void	
+run() : void	

58. ábra

Az információs panelben megjelenő időzítő.

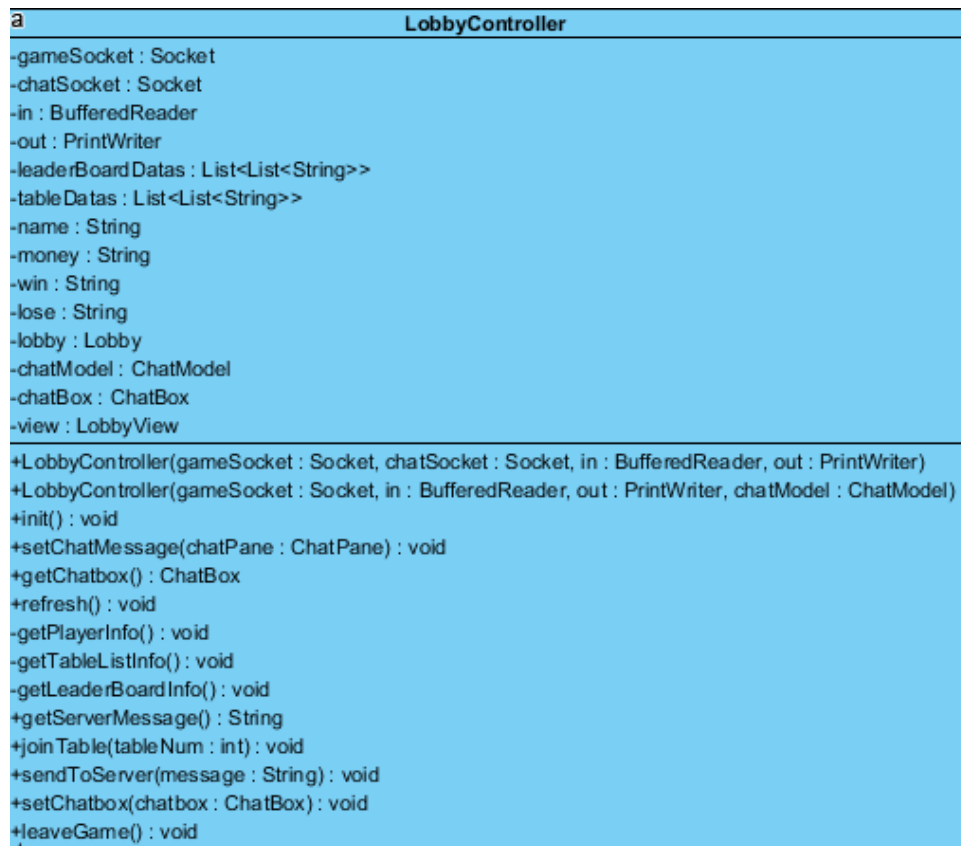
### CardFactory osztály

a	<b>CardFactory</b>
-url : String	
+CardFactory(url : String)	
-init() : void	

59. ábra

Ez az osztály felel a kártya grafikus előállításáért, a konstruktor kap egy URL-t és az alapján megkeresi a laphoz tartozó png-fájlt.

### LobbyController osztály



60. ábra

Ez az osztály gyűjti össze az adatokat Lobby felállításához.

### Fontosabb Adattagok:

- List<String> datas: Játékos adatai
- List<List<String>> leaderBoardDatas: Leaderboard adatai
- List<List<String>> tableDatas: Asztalok adatai
- String name,money,win,lose: A játékos adattáblában tárolt adatai
- final ChatModel chatModel
- ChatBox chatbox

Két konstruktora van az első LobbyController(Socket gameSocket, Socket chatSocket, BufferedReader in, PrintWriter out) akkor hívódik meg, amikor a játékos bejelentkezik.

A második LobbyController(Socket gameSocket, BufferedReader in, PrintWriter out, ChatModel chatModel) amikor a játékos kilép a játékból.

Fontosabb Eljárások:

- setChatMessage: Chatablak hozzáadása.
- refresh(): Adatok frissítése.
- getPlayerInfo(): Adatok lekérdezése.
- getTableListInfo(): A táblaadatok frissítése.
- getLeaderBoardInfo(): A ranglista adatok frissítése..
- getServerMessage(): Szerverüzenet.
- joinTable(int tableNum): Csatlakozás játékasztalhoz.
- sendToServer(String message): A szerverüzenet küldése.
- leaveGame(): Játék elhagyása.

ClientModel osztály

a	ClientModel
-in : BufferedReader	
-out : PrintWriter	
-playersPanel : PlayerBox	
+ClientModel(in : BufferedReader, out : PrintWriter)	
+getServerMessage() : String	
+sendClientMessage(message : String) : void	
+addHandPanel(playerId : String, index : String) : void	
+removeHandPanel(playerId : String, index : String) : void	
+getHand(playerId : String, index : String) : ObservableList	
+getPlayerBox(playerId : String) : PlayerBox	
+addPlayerBox(id : String, username : String, money : String) : void	
+removePlayer(id : String) : void	

61. ábra

Ez az osztály tárolja a játékosok adatait és ez alapján frissíti a GUI-t a controller-osztály.

Fontosabb Adattagok:

- LinkedList<PlayerBox> playersPanel: itt tárolódnak a játékosok

Fontosabb Függvények:

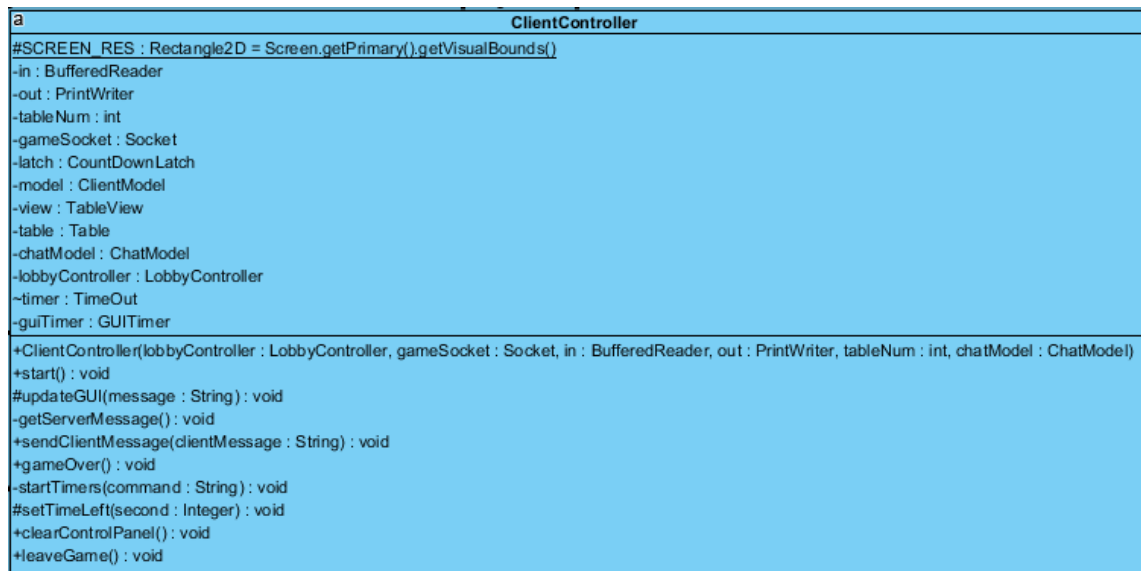
- String getServerMessage(): A szerverüzenet fogadás.
- ObservableList getHand(String playerId, String index): Az adott játékos adott kezével tér vissza.

- PlayerBox getPlayerBox(String playerId): A játékost szimbolizáló panellel tér vissza.

#### Fontosabb Eljárások:

- sendClientMessage(String message): Válasz küldése.
- addHandPanel(String playerId,String index): Játékos kezét szimbolizáló panel hozzáadása.
- removeHandPanel (String playerId, String index): Kezet szimbolizáló panel eltávolítása.
- addPlayerBox(String id,String username,String money): Játékost szimbolizáló panel hozzáadása.
- removePlayer(String id): A játékost szimbolizáló panel eltávolítása.

#### ClientController osztály



62. ábra

Ez osztály felelős a modell és GUI közötti kommunikációért, illetve a szerverüzenetek feldolgozásáért.

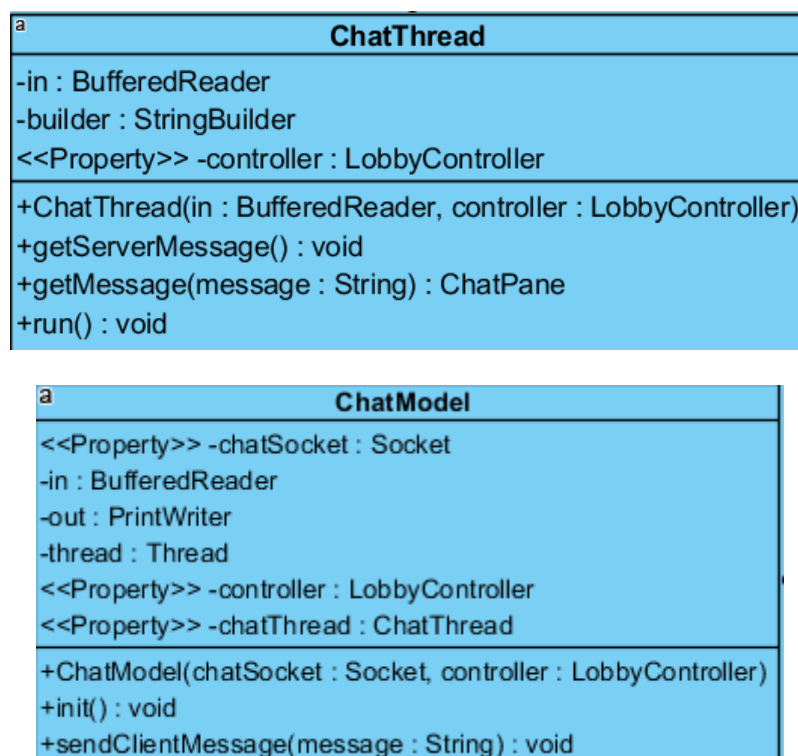
#### Fontosabb Adattagok:

- ClientModel: Model
- TableView: View
- final int tableNum: Asztal sorszáma
- Table table: Asztal
- ChatModel: ChatModel
- final Socket: GameSocket
- final LobbyController: LobbyController
- CountdownLatch latch: A játék végén nullázódó latch.
- Timeout timer: Minden választási lehetőségnél elinduló időzítő.
- GUITimer guiTimer: A GUI-n megjelenő időzítő.

#### Fontosabb Eljárások:

- updateGUI (String message): A GUI manipulációjáért felelős függvény az üzenet alapján.
- getServerMessage(): Szerverparancsok feldolgozása és a GUI módosítását meghívó eljárás.
- sendClientMessage(String clientMessage): A válasz küldése a szervernek.Időzítő leállítása.
- gameOver():A játék végén lefutó függvény, visszairányít a lobbyba.
- leaveGame():A játékból való kilépéskor lefutó függvény.
- startTimers(String command): Ez az eljárás indítja el az időzítőket.

#### ChatThread és ChatModel osztály



63. ábra

Ez a két osztály felelős a chat működésért alapjában megegyezik a játékszerver működésével csak annyi funkcionális különbséggel, hogy egyetlen feladat a kliensek közti szöveges kommunikáció.

## Tesztelés

### Bejelentkezés

<u>Esemény</u>	<u>Várt eredmény</u>
csatlakozás localhost: 4444,4445 portra (nem fut a szerver a megadott helyen)	Hiba üzenet: nem található a szerver
csatlakozás localhost: 4444,4445 portra(fut a szerver a megadott helyen)	Bejelentkezési ablak
Bejelentkezés nem létező felhasználóval	Hiba nem létezik a felhasználó
Bejelentkezés rossz jelszóval	Hiba rossz jelszó
Regisztráció helyes adatokkal	Üzenet jelentkeztek be (adatbázis frissült)
Regisztráció helytelen adatokkal (végig próbálva üresmezőkre illetve nem egyező jelszó vagy email párosra)	hiba írja be a hiányzó mezőt vagy a két cella nem egyezik
Regisztrációval már létező felhasználónévvel majd már létező e-mail címmel	Mindkét esetben hiba, már létezik a felhasználónév vagy az e-mail cím
Bejelentkezés már online felhasználóval	Hiba valaki már bejelentkezett
Előző esett után kilépés a játékból, majd újra próbálkozás	Lobbyba jutunk
Regisztrációs felületen a vissza felíratra kattintás	Visszalép a bejelentkezésre.

## Lobby

<u>Esemény</u>	<u>Várt eredmény</u>
Chat üzenet	megjelenik az üzenet
Leaderboard fejlécének végig kattintása	a táblázat módosul az oszlopnév szerint
csatlakozás olyan asztalhoz amihez nincs elég pénze a játékosnak	nincs elég pénze üzenet
Asztal kiválasztása majd Ok gomb megnyomása	Csatlakozás a játékhoz
Asztal kiválasztása majd Ok gomb megnyomása valaki játszik közben	Üzenet valaki játszik majd kör végén Csatlakozás a játékhoz, a másik játékosnak megjelenik az új játékos és minden folytatódik.A csatlakozó játékosnak megjelenik az ablakon lévő játékos
Egyszerre csatlakozás az asztalhoz 3 új játékos	Először NullPointerExceptiont jelzett a program kliens oldalon nem találta a játékos paneljét a Gui.Megoldott a hibaforrása az volt, hogy a várakozók csak a játékost értesítették, egymást nem. Ezután minden játékos látta egymást.

## Játék

### 2 vagy több játékosnál

<u>Esemény</u>	<u>Várt eredmény</u>
Tét megtétele	Vár a következő játékosra utána lapokat oszt a szerver ezután hit, stand double down ha nincs blackjack-je a játékosnak
Ha ő a soron következő játékos játék hit/stand opció választása	Ha túlmegy a 21-en vagy megáll a következő játékos következik vagy az osztó
Osztó lapjai láthatóvá válnak	Játék vége, ki nyert, pénz kiutalása, folytatás panel
egyik játékos nem folytatja	Eltűnik az asztalról
Egyik játékos kör elején kilép	A szerver végig játssza a játékot
Játék vége	Mindkét játékos adatai frissültek az adatbázisban
Kilépés játék közben	A szerver lejátssza helyette a játékot, adatok frissülnek az adatbázisban
Tét tévése	Lapok után hit, stand, vagy double down opció,összeg levonása
Double down opció	Egy lapot kap és dupázódik a játékos tétje
2 játékos 2 különböző asztalon játszik	Minden megfelelően működik,nincs szerver probléma
Egy körben több játékos elhagyja a játékot	A játékosok listáján az egyik Player osztály üzenetet szeretet volna küldeni,miközben egy játékos eltávolította magát ez egy ConcurrentModificationException kivételt generált. A kivétel kezelve lett, ezután a probléma már nem állt fent.



Valamilyen opció választása	A számláló leáll, új opciónál újraindul
Belépés a játékba, majd nem választottam a lehetőségek közül.	15 másodperc múlva feltette a minimális tétet a szerver, a lapok megkapása után az én körömben 15 másodperc után megállt, a kör végén 15 másodperc után ledobott az asztalról a szerver.

## Chat

<u>Esemény</u>	<u>Várt eredmény</u>
Belépéskor Lobbyban chat üzenet	Mindkét játékosnak megjelenik
Játék megkezdése után üzenet	Mindkét játékosnak megjelenik

## Fejlesztési lehetőségek

A játék során nem sikerült implementálnom a split hand funkciót, ami egy olyan lépés, ami akkor válik elérhetővé ha a játékos első kétlapjának mintája megegyezik(pl.: Ász-Ász), ekkor ketté oszthatja a lapjait és két kezére külön játszhat. A másik ilyen funkció, ami nem minden helyen elfogadott a Surrender opció ez könnyebben implementálható, ennek az opciónak a kijátszásával az első két lap után feladhatjuk a játékot ilyenkor a megtett tét felét visszakapjuk és a maradék a bankhoz kerül.

A játék során feltételezzük, hogy ha valaki játszik ő vagy a kör végén kiszáll vagy kilép a játékból, de semmiképp nem hagyja ott a játékot nyitott ablaknál, mert a többi játékos ekkor vár rá, hogy megtegye a lépését, ezt kiküszöbölendő egy időzítőt lehetne implementálni a szerver oldalra, ami egyszerűen tovább lépteti a játékot. Ezen felül különböző kártyajátékokat is lehetne implementálni a játékba kiterjesztve a programot egy Casino-vá illetve webesfelületre átültetni a játékot.

A grafikus felületen lehetne animációkat alkalmazni az osztásra, illetve realisztikusabban ábrázolni az asztalt például zsetonok használatával.

## Hivatkozások

<https://hu.wikipedia.org/wiki/Huszonegy>

1.Játék szabályok

Elérhető: 2019.május

<https://www.callicoder.com/javafx-registration-form-gui-tutorial/>

2.Creating a registration form in JavaFX

Elérhető: 2019.május

<https://people.inf.elte.hu/mozsik/2016172/progtech2/>

3.Mócsi Krisztián honlapja

Elérhető: 2019.május

<http://kitlei.web.elte.hu/>

4.Kitlei Róbert honlapja

Elérhető: 2019.május

<https://www.apachefriends.org/hu/index.html>

5.XAMMP letöltési hely, illetve telepítési útmutató

Elérhető: 2019.május

<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>

6.Model–view–controller

Elérhető: 2019.május

[www.mindrot.org](http://www.mindrot.org)

7.BCrypt

Elérhető: 2019.május

<https://stackoverflow.com/>

8.

Elérhető: 2019.május

<https://github.com/>

9.

Elérhető: 2019.május

<https://github.com/teddyteh/Multiplayer-Card-Game>

10.Multiplayer-Card-Game by teddyteh

Elérhető: 2019.május

<https://gluonhq.com/products/scene-builder/>

11.Scene Builder alkalmazás oldala

Elérhető: 2019.május