Suunnitteludokumentti

pokeriv3

Helsinki 8.4.2008

Ohjelmistotuotantoprojekti

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Kurssi

581260 Ohjelmistotuotantoprojekti (6 ov)

Projektiryhmä

Anne-Marie Grönroos
Markus Oksanen
Ville Pulkkinen
Tommi Sankola
Lari Sorvo

Asiakas

Teemu Saukonoja
Tomi Pasanen

Ryhmän ohjaaja

Kim Ervasti

Johtoryhmä

Kimmo Simola

Kotisivu

`http://www.cs.helsinki.fi/group/pokeriv3/`

Versiohistoria

Versio	Päiväys	Tehdyt muutokset
0.1	18.2.2008	Ensimmäinen versio
0.8	27.3.2008	Katselmoitava versio
0.9	31.3.2008	Tarkastettava versio
0.95	3.4.2008	Korjattu toiseen tarkastukseen
1.0	8.4.2008	Jäädytetty versio

Sisältö

1 Johdanto

1.1 Dokumentin tarkoitus

Tämä on Pokeriv3 -ryhmän suunnitteludokumentti pokeripalvelinohjelmiston toteutukselle. Suunnitteludokumentin tarkoitus on kertoa, miten vaatimusmäärittelydokumentista ilmi tulleet vaatimukset tullaan toteuttamaan. Suunnitteludokumentissa kuvataan järjestelmän arkkitehtuuri, komponentit, tietokanta sekä luokat. Suunnitteludokumentissa on kuvattu myös osien väliset rajapinnat. Komponenttien tarkempi toiminta tietorakenteiden ja algoritmien osalta on dokumentoitu vain, mikäli se on nähty tarpeelliseksi.

1.2 Aiheen kuvaus

Kehitettävä pelipalvelinohjelmisto tarjoaa rajapinnan, jonka avulla voidaan testata erilaisia tekoälyratkaisuja peluuttamalla niitä eri tasoisia ja tyylisiä tietokone- ja ihmisvastustajia vastaan. Ohjelmisto mahdollistaa eräajon, jolloin voidaan pelata suuri määrä käsiä ja täten saada luotettavampaa tietoa tekoälyratkaisun ”hyvyydestä” minimoimalla sattuman osuus korttien jakautumisessa. Järjestelmä mahdollistaa pokerin lisäksi myös muiden vuoropohjaisten pelien pelaamisen. Pelimoduulit voivat tarjota käyttäjälle mahdollisuuden muokata pelin sääntöjä tai parametrejä. Projektin yhteydessä on tarkoitus kehittää järjestelmään pokeripeli ”Texas Hold’em”-säännöillä.

2 Sanasto

Järjestelmä Komponenteista ja niiden välisistä suhteista koostuva itsenäinen yksikkö, joka tuottaa erilaisia palveluita käyttäjälle. Pokeriv3 ohjelmiston yhteydessä järjestelmällä tarkoitetaan koko Pokeriv3-ohjelmistoa.

Osajärjestelmä Järjestelmän tapaan itsenäinen kokonaisuus, mutta osajärjestelmä on osa suurempaa järjestelmää. Pokeriv3 ohjelmiston yhteydessä osajärjestelmällä tarkoitetaan esimerkiksi palvelinta tai asiakasohjelmaa.

Komponentti Osajärjestelmän osa, joka tuottaa yhteenkuuluvia palveluita järjestelmälle. Pokeriv3 ohjelmiston yhteydessä komponentilla tarkoitetaan esimerkiksi palvelukokonaisuuksia, jotka liittyvät tietokannan käsittelyyn.

Moduuli Itsenäinen osajärjestelmä, joka voidaan liittää sen hyvinmääriteltyjen rajapintojen avulla tiettyyn järjestelmään. Moduuli voidaan liittää järjestelmään ja poistaa siitä. Pokeriv3 ohjelmiston yhteydessä moduulilla tarkoitetaan pelimoduulien asiakaskomponenttia ja sen palvelinkomponenttia, sekä bottia ja käyttöliittymäkomponentteja.

Luokka Luokka on jonkin reaalimaailmaan rinnastettavien ominaisuuksien ja toimintojen kokonaisuus. Se on suunnitteludokumentin matalimman tason ohjelmistoyksikkö. Tätä tarkempaan kuvaukseen suunnitteludokumentissa ei yleisesti ottaen mennä.

Pelimoduuli Järjestelmään liitettävä moduuli, joka sisältää tietyn pelin toimintalogiikan ja sen käyttöön tarvittavat komponentit.

Pelimoduulin palvelinkomponentti on pelimoduulin osa, joka sijoittuu järjestelmässä palvelimen koneelle. Siellä sijaitsee varsinainen pelimoottori, joka toteuttaa pelissä toimivan logiikan.

Pelimoduulin asiakaskomponentti on pelimoduulin osa, joka ladataan peliin liityttäessä asiakkaan koneelle palvelimelta. Se sisältää pelimoduulin käyttöliittymän sekä siihen liittyvien toimintojen tulkkaamisen pelimoduulin palvelinkomponentille välitettävään muotoon.

Viestitysolio on kiertoilmaus *CommunicationObject*-oliolle, joita käytetään tiedonvälitykseen järjestelmässä.

Viestittäjä on yleisnimitys komponentille, joka osallistuu järjestelmän viestinvälitykseen.

Skeema on looginen kokonaisuus tietokannassa. Pokeriv3-projektissa niitä käytetään eri pelimoduulien tarvitsemien tietokantaobjektien erottelemiseksi itsenäisiksi kokonaisuuksikseen.

Rooli on tietokannassa joukon oikeuksia yhteen niputtava käyttäjätaso, joka voidaan antaa käyttäjälle väliaikaisesti tai pysyvästi.

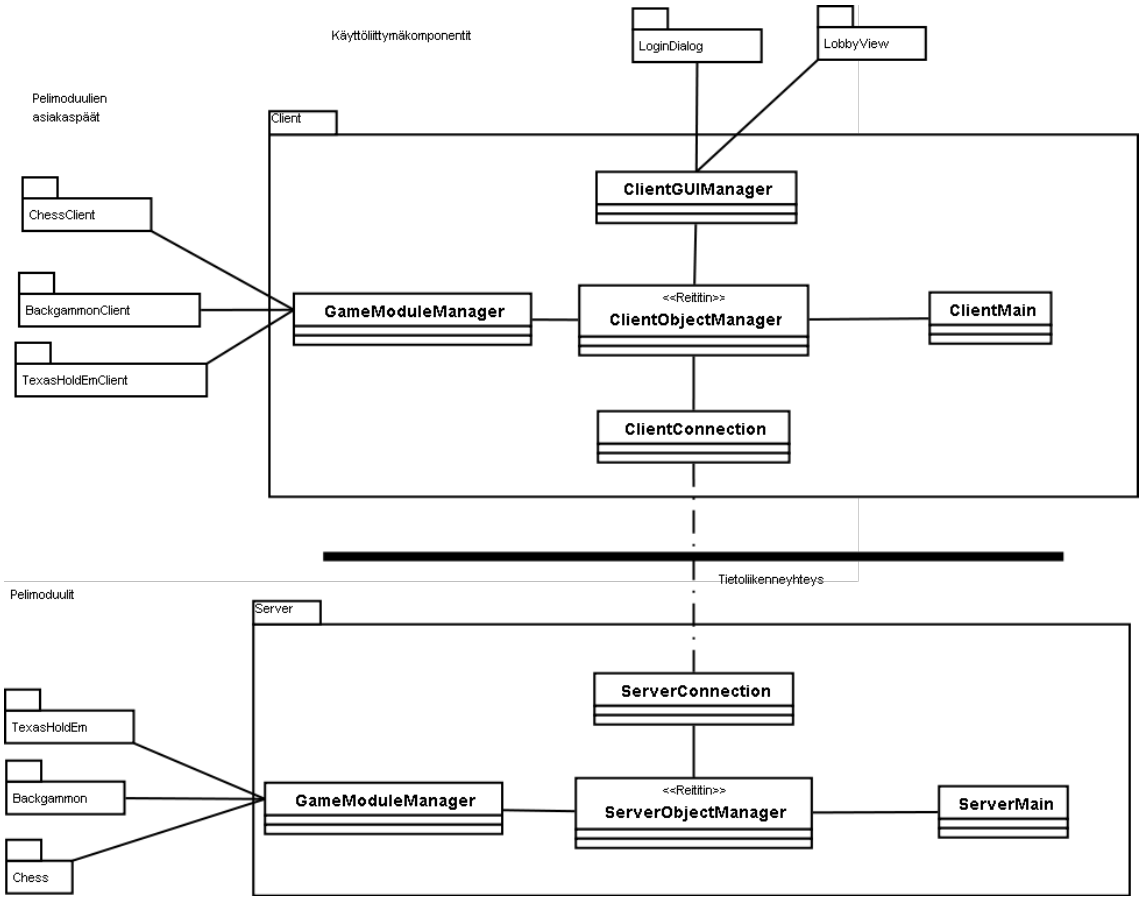
Meerkat API on Poker Academyn määrittelemä rajapinta Texas Holdem -pokerin pelaamiseen. Florida-ohjelmiston Texas Holdem -toteutus perustuu samaan rajapintaan.

ANSI SQL-92 on tietokantakielistandardi, jonka kanssa toteutettava Florida-ohjelmisto on yhteensopiva. Näin järjestelmän toimivuus ei ole riippuvainen tietystä tietokantajärjestelmästä, vaan mikä tahansa ANSI SQL-92 -yhteensopiva tietokantajärjestelmä käy.

3 Arkkitehtuurisuunnitelma

3.1 Järjestelmän kokonaiskuvaus

Järjestelmä koostuu kahdesta varsinaisesta osajärjestelmästä: palvelimesta ja asiakasohjelmasta. Lisäksi palvelimeen liitetään pelimoduuli, jonka ns. pelimoottori toimii pelimoduulin palvelinkomponentissa. Käyttöliittymä ja pelaajan toiminnallisuuteen liittyvät



Kuva 1: Yleiskuva toiminnallisista komponenteista

tapahtumien tulkkaus toteutetaan pelimoduulin asiakaskomponentissa. Pelimoduulin asiakaskomponentti ladataan asiakasohjelmaan palvelimelta. Järjestelmä tallentaa myös joka jaon jälkeen tietoja palvelimeen liitettävään tietokantaan. Järjestelmä tullaan toteuttamaan siten, että se toimii PostgreSQL -tietokantajärjestelmän kanssa. Mikä tahansa ANSI SQL-92 -yhteensopiva tietokantajärjestemä sopii järjestelmän kanssa käytettäväksi.

Järjestelmän toteutuskielenä käytetään Javaa. Käyttäjä käyttää järjestelmää pääosin graafisen käyttöliittymän kautta, mutta joitain toiminnallisuuksia tullaan toteuttamaan myös siten, että ne voidaan toteuttaa komentoriviltä syöttäen (esim. palvelimen pystytys ja eräajo).

Järjestelmän päätarkoitus on tarjota alusta bottien testaamiseen ja näin ollen asiakasohjelmaan voidaan liittää botti, joka käyttää samaa rajapintaa kuin normaali käyttäjä. Asiakkaan pyynnöstä käyttäjän ja botin rajapinta asiakasohjelmassa pyritään toteuttamaan Poker Academyn Meerkat-APIa hyväksikäyttäen.

3.2 Yleiskäyttöinen palvelin

Palvelimen tehtävä on hoitaa kaikkea eri moduulien ja asiakasohjelman välistä viestintää. Palvelin on yhteydessä asiakasohjelmiin ja hoitaa kaikki yhteydenpitoon ja viestintään liittyvän hallinnoinnin. Palvelimen toteutus ei ota kantaa siihen liitettyjen pelimoduulien toteutuksesta, vaan tarjoaa ainoastaan yhteiset rajapinnat tiedonvälitykseen.

Palvelin on myös vastuussa kaikista pyydetyistä tietokantaoperaatioista. Palvelin tarjoaa olion, joka suorittaa tietokantaoperaatioita.

Tietoliikenteen salaus on myös osa palvelimen keskeistä toiminnallisuutta. Palvelin tarjoaa kaksi yhteyttä asiakasohjelmaa kohden: salaamattoman ja salatun yhteyden. Salattua yhteyttä käytetään arkaluontoisen datan toimittamiseen verkon yli. Tällaisia ovat esimerkiksi pelaajan käyttäjätunnus ja salasana, sekä moduulien erikseen salattavaksi pyydetyt tiedot.

3.2.1 Palvelimen käyttöliittymä

Palvelimen käyttöliittymä on yksinkertaisesti komentorivi, josta se voidaan käynnistää tietyin asetuksin. Sille toteutetaan myös graafinen käyttöliittymä, mikäli siihen jää aikaa.

3.3 Yleiskäyttöinen asiakasohjelma

Asiakasohjelman tehtävänä on luoda yhteys asiakkaan ja palvelimen välille, ylläpitää yhteyttä, välittää viestitysolioita verkon yli palvelimelle sekä ohjata objekteja oikeille moduuleille asiakasohjelmassa. Asiakasohjelma myös vastaanottaa ClientObject -tyyppisiä viestitysolioita. Lisäksi asiakasohjelma antaa käyttäjälle mahdollisuuden luoda pelin tai liittyä peliin. Halutun pelin komponentit lähetetään palvelimelta asiakasohjelmaan vasta peliin liityttäessä.

Asiakasohjelma koostuu tehtävien mukaan jaotelluista komponenteista. Pääohjelman tarkoitus on toimia eri komponenttien toiminnot kokoavana yksikkönä. Yhteydenluonti-komponentin tehtävä on luoda normaali sekä salattu yhteys palvelimeen. Yhteyksiä käyttää ClientObjectManager-komponentti, joka toimii yhteydestä tulevien ja sinne menevien viestitysolioiden reitittäjänä. Sen tehtävänä on olioiden reititys asiakasohjelman komponenteille.

3.3.1 Asiakasohjelman käyttöliittymät

Asiakasohjelmaan kuuluu palvelimen sille toteuttamia käyttöliittymiä sekä pelimoduulikohtaisia käyttöliittymiä ja käyttöliittymäkomponentteja, jotka lisätään asiakasohjelmaan peliin liityttäessä.

Yhteydenmuodostus toteuttaa yhteydenmuodostukseen liittyvät syötekentät kuten palvelimen osoitteen ja portin ja painikkeen yhteyden muodostamiseen. Tämä on ainut ikkuna, joka on toteutettu kiinteästi asiakasohjelmaan.

Sisäänkirjautuminen toteuttaa sisäänkirjautumiseen liittyvät syötekentät kuten käyttäjätunnuksen ja salasanan. Ikkunassa on painike "Log in", joka toteuttaa sisäänkirjautumisen. Lisäksi ikkunassa voidaan painaa painiketta "Sign in as a new user", jolloin ohjelma yrittää lisätä käyttäjän uudeksi käyttäjäksi annetuilla tunnuksella ja salasanalla. Salasana pitää tässä tapauksessa antaa toisen kerran varmistukseksi oikeinkirjoituksesta.

Aula on ikkuna, jossa on nähtävillä kaikki palvelimella sillä hetkellä pyörivät pelipöydät. Pelipöydän riviä klikkaamalla listan oikealle puolelle tulee tarkemmat tiedot pelipöydästä, sekä siihen liittyvät syötekentät. Tarkemmat tiedot pelipöydästä ja peliin liittymiseen liittyvät syötekentät tulevat *JFrame*-oliona pelipalvelimelta, jolloin jokaisen pelimoduulin pelitiedot ja syötekentät ovat yksilöllisiä. Pelilista-ikkunassa on napit peliin liittymiselle ja uuden pelipöydän luomiselle. Tässä näkymässä on pelaajan myös mahdollisuus liittää asiakasohjelmaan botti, joka pelaa pelejä ihmispelaajan sijasta.

Asetukset -ikkunassa voidaan muuttaa pelaajakohtaisia asetuksia. Täältä voidaan ladata botti pelaamaan ihmiskäyttäjän puolesta.

Pelitietojen selaus -ikkunassa voidaan tehdä kysely, jonka jälkeen se näyttää listan jaoista. Jos käyttäjä on "superuser", voi se tehdä sille tarkoitettuja lisäkyselyitä. Kyselyt mukana vaatimusmäärittelydokumentin liitteessä 3.

3.4 Pelimoduuli

Pelimoduuli määrittelee pelisäännöt, pelin kulun ja toteutukseen ja pelaamiseen vaaditut komponentit, esim. korttipakan ja pelaajat. Pelimoduuli vastaa myös käyttöliittymäkom-

ponenttien ja toimintojen tulkkaukseen liittyvien komponenttien tarjoamisesta asiakasohjelmalle mukaanlukien mahdolliset tietokantakyselyt. Pelimoduulin erottaminen palvelimesta mahdollistaa uusien pelien lisäämisen järjestelmään, kunhan lisättävä peli toteuttaa pelimoduuleille yhteisesti määritellyt rajapinnat. Projektin puitteissa toteutetaan pelimoduuli tavallisella 52:n kortin pakalla pelattavalle Texas Hold 'em -pokerille. Asiakkaan toivomuksesta Texas Hold 'em -pokerin toteuttamisessa on käytetty Poker Academyn Meerkat API:a. Käytetty versio on 2.5.

3.4.1 Pelimoduulin palvelinkomponentti

Pelimoduulin palvelinkomponentti on eräänlainen pelimoottori, joka määrittelee pelisäännöt, pelin kulun ja toteutukseen ja pelaamiseen vaaditut komponentit. Palvelimelle luodaan pelipöytiä luomalla pelimoottorista ilmentymiä. Pelimoottori on itsenäinen kokonaisuus, joka vastaa pelin etenemisestä saaden syötteitä pelaajilta.

3.4.2 Pelimoduulin asiakasohjelmakomponentti

Palvelimen yhteydessä suoritettava pelimoduulin osa lähettää asiakkaalle graafisessa käyttöliittymässä näytettävät pelitapahtumia ja -tilannetta kuvaavat sekä pelin toimintojen käyttämiseen tarkoitetut käyttöliittymäkomponentit. Näin käyttäjä näkee käyttöliittymässään esim. pelipöydän, korttien kuvat, pelimerkit käytettävissään olevien toimintojen painikkeet. Lähetettävät komponentit ja niiden tarkat sisällöt riippuvat pelimoduulissa tehdyistä määrityksistä. Pelimoduulin asiakasohjelmakomponentti käsittelee pelaajan tapahtumat ja paketoi ne pelimoduulin palvelinkomponentille lähetettävään muotoon. Se osaa myös purkaa ja käsitellä pelimoottorilta sille tulleet objektit.

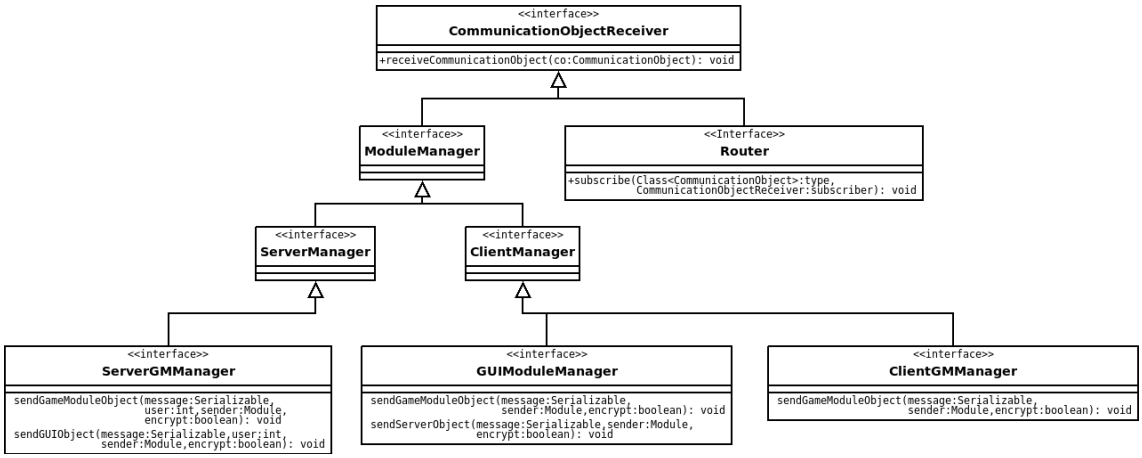
3.5 Tietokanta

Palvelimen käyttäjätietokanta sisältää käyttäjien tunnukset ja salasanat. Lisäksi jokaisella palvelimeen liitettävällä pelimoduulilla voi olla tietokannassa oma skeemansa, joka on samassa tietokannassa kuin palvelimen oletusarvoinen public-skeema. Skeemoja käyttämällä kaikki pelimoduulit voidaan liittää samaan tietokantaan ilman tietokantaobjektien nimien päällekkäisyyksistä aiheutuvia ongelmia. Lisäksi rooleja käyttämällä voidaan rajata käyttäjän pääsy vain tiettyihin skeemoihin ja näin estää pelimoduuleja muuttamasta toistensa tietoja. Roolin voi asettaa käyttäjälle vain väliaikaisesti ja palauttaa käyttäjän oikeustaso entiselleen tietokantaoperaation päätyttyä.

4 Rajapinnat

4.1 Viestitysolioiden käsittelyyn liittyvät rajapinnat

Kuvan 2 rajapinnat toteutetaan palvelin- ja asiakasohjelmassa.



Kuva 2: Viestittäjien hierarkia

4.1.1 Interface CommunicationObjectReceiver

Tämän rajapinnan toteuttaa luokka, joka ottaa vastaan *CommunicationObject*-olioita.

receiveCommunicationObject(CommunicationObject co)
Ottaa vastaan *CommunicationObject*-olion ja käsittelee sen.

4.1.2 Interface Router extends CommunicationObjectReceiver

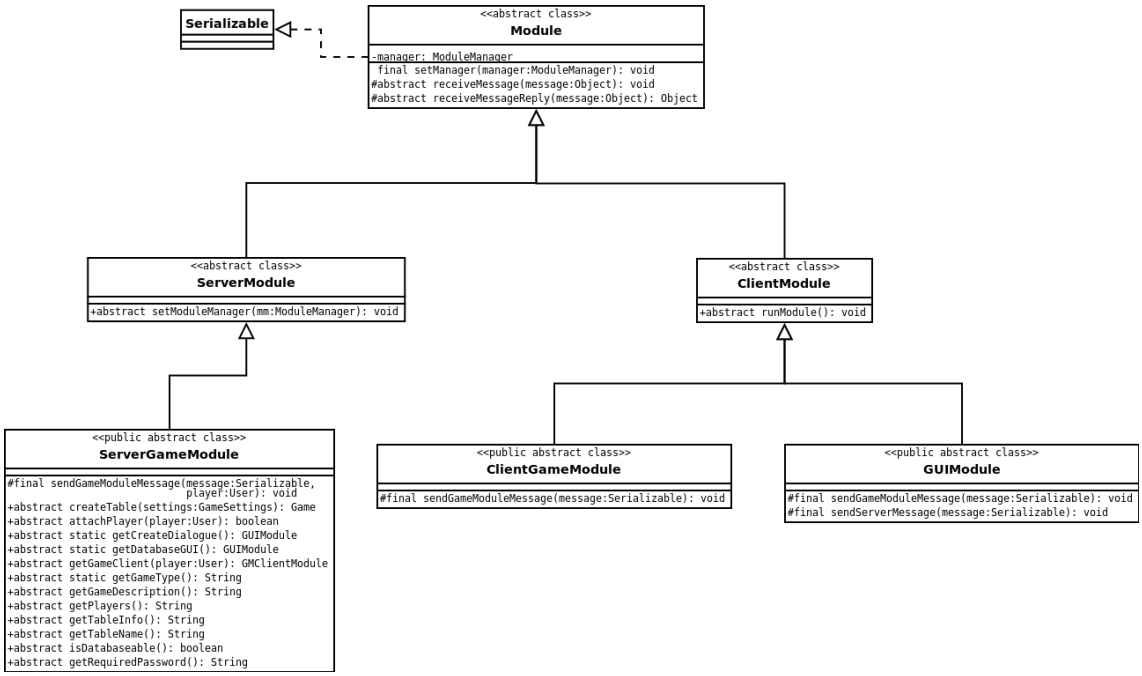
Tämän rajapinnan toteuttaa luokka, joka toimii *CommunicationObject*-olioiden reitittäjänä.

subscribe(Class<CommunicationObject> type, CommunicationObjectReceiver subscriber)
Pyytää *Router*-oliota lähettämään saamansa *type*-tyyppiset *CommunicationObject*-oliot *subscriber*-oliolle.

4.2 Moduulien rajapinnat

4.2.1 Moduulit

Moduulien rajapintahierarkia näkyy kuvassa 3. Ulkoisen moduulin tulee periä jokin abstrakteista luokista *ServerGameModule*, *ClientGameModule* tai *GUIModule*. *ServerGameModule*n perivät pelimoduulin palvelinkomponentit, *ClientGameModule*n perivät pelimoduulin asiakaskomponentit ja *GUIModule*n perivät käyttöliittymäkomponentit, jotka eivät suoraan liity mihinkään pelin ilmentymään. Varsinainen pelinäkymä (ks. kuvaliite) tuotetaan *ClientGameModule*ssa.



Kuva 3: Moduulin rajapinnat

4.3 Palvelimen rajapinnat

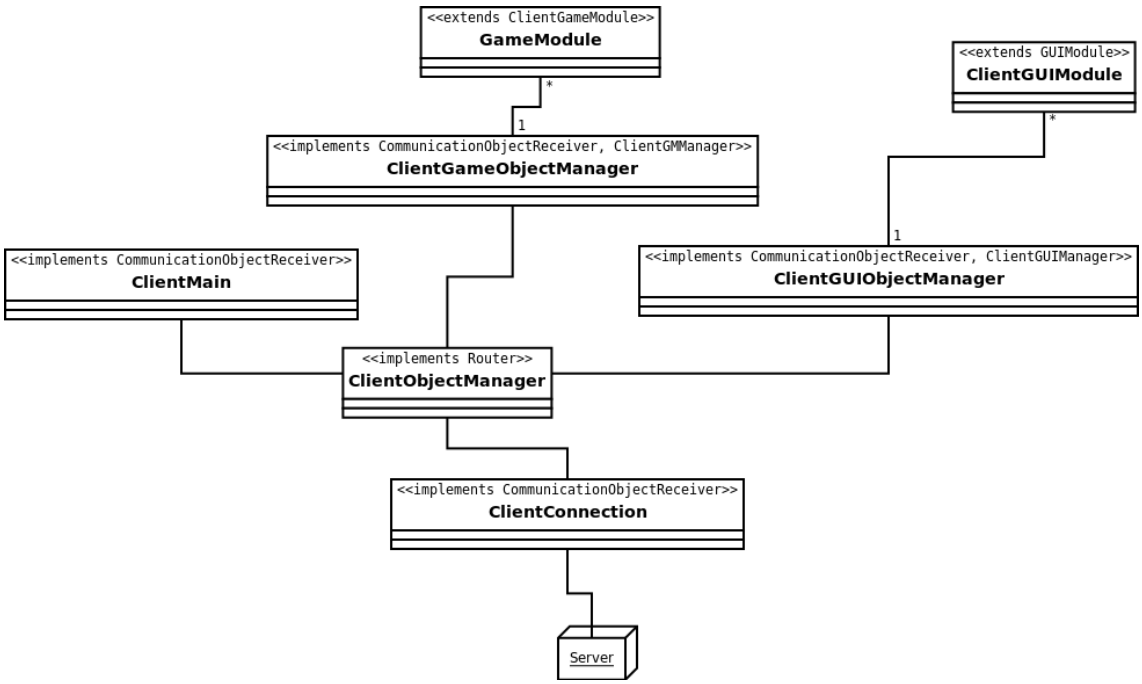
Palvelin toimii yhteyssolmuna kolmelle eri komponentille: asiakasohjelmalle, pelimoduulin palvelimen yhteydessä suoritettavalle osalle ja tietokannalle. Tämän lisäksi palvelin tarjoaa rajapinnan tietokannan käyttöön. Tietokantaa käyttävät pelimoduulin palvelimen yhteydessä suoritettava osa ja palvelimen sisäinen käyttäjähallinta.

4.4 Asiakasohjelman rajapinnat

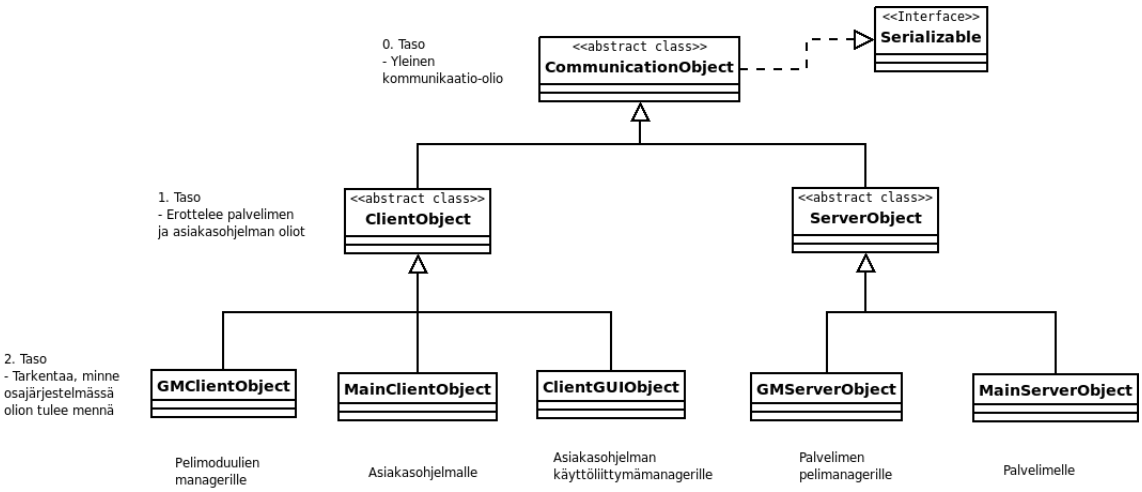
Asiakasohjelma toimii yhteyssolmuna kolmelle eri komponentille: palvelimelle, pelimoduulin asiakasohjelman yhteydessä suoritettavalle osalle ja käyttöliittymälle. Asiakasohjelma vastaa tietoliikenteestä palvelimen ja pelimoduulin sekä käyttöliittymän välillä. Tässä tapauksessa käyttöliittymällä tarkoitetaan niitä käyttöliittymiä, jotka liittyvät vain asiakasohjelman käyttöön. Pelimoduulin asiakasohjelman yhteydessä suoritettavalla osalla on olemassa näistä käyttöliittymistä erilliset käyttöliittymät.

5 Viestitysoliot

Tieto asiakasohjelman ja palvelimen välillä liikkuu kapseloituna *CommunicationObject*-olioihin. Asiakasohjelma käyttää viestitysolioita palvelimen tarjoamien palvelujen käyt-



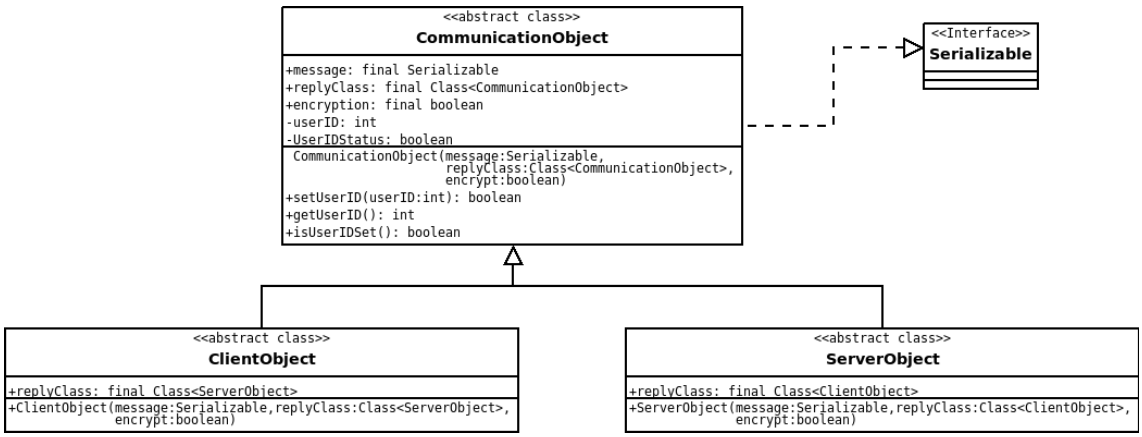
Kuva 4: Asiakasohjelman luokkien rajapintojen toteutus



Kuva 5: Viestitysolioiden hierarkia

tämiseen, ja palvelin käyttää olioita välittääkseen tiedot asiakasohjelmalle. *Router*-rajapinnan toteuttava komponentti huolehtii pakettien reitityksestä osajärjestelmän sisällä, välittäen ne tyypin mukaan sille komponentille, joka on kunkin tyyppisen *CommunicationObject*-olion tilannut. Kuvassa 5 näkyy viestysolioiden hierarkia. *CommunicationObject* on yläluokka kaikille viestitysolioille. Seuraavalla tasolla erotellaan palvelimelle ja asiakasohjelmalle osoitetut viestit. Ja näiden aliluokat identifioivat sen komponentin palvelimen tai asiakasohjelman sisällä, jolle viesti kuuluu. 1. tason viestitysolioita perimällä voidaan järjestelmään tuoda uusia komponentteja, jotka keskustelevat palvelimen kanssa. Lisäksi kaikki viestitysoliot toteuttavat *Serializable*-rajapinnan, jotta niitä voidaan lähettää tietoliikenneyhteyden käyttämän *ObjectStream*-in välityksellä.

5.1 Abstraktit luokat



Kuva 6: Viestitysolioiden abstraktit luokat

Viestitysolioiden abstraktit luokat määrittävät kaikille viestitysolioille yhteisten toimintojen toiminnallisuuden.

CommunicationObject -luokka on kaikkien viestitysolioiden yliluokka. Se määrittelee viestitysluokkien toiminnallisuuden sen suhteen, mitä niillä voidaan kuljettaa, miten viestitysolio yksilöidään palvelimella, ja kenelle mahdollinen vastausviesti pitää lähettää. *CommunicationObject*-luokasta ei voi tehdä ilmentymiä.

message -kenttä on viestitysolion viestikenttä. Viestitysolioiden välityksellä voi välittää siis minkälaisia olioita tahansa.

replyClass -kenttä antaa *CommunicationObject*-luokan, josta tehtyyn ilmentymään vastaus alkuperäiseen viestiin tulee tehdä.

encryption -kenttä kertoo, pitääkö viesti lähettää salatun yhteyden kautta.

userID -kenttä yksilöi viestitysolion palvelimen puolella johonkin tiettyyn käyttäjään.

userIDStatus -kenttä on totuusmuuttuja, joka kertoo, onko userID asetettu vai ei. Tämä on oletusarvoisesti epätosi.

communicationObject -konstruktorin parametrit asettavat arvot vakiokentille *message*, *replyClass* ja *encryption*.

isUserIDSet -metodi tarkistaa, onko userID jo asetettu.

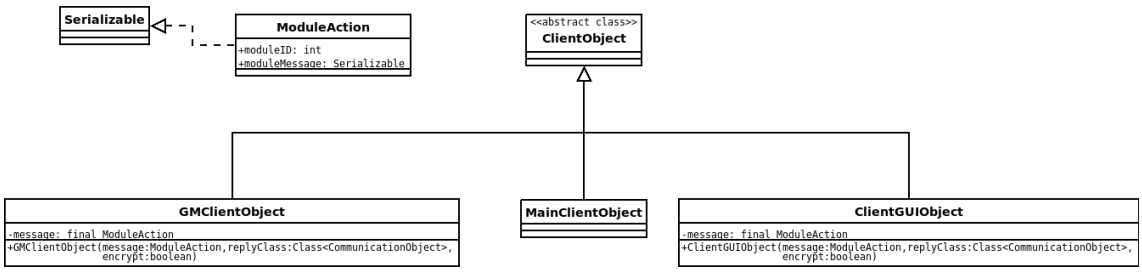
setUserID -metodi sijoittaa userID-kentälle arvon, jos sillä ei vielä arvoa ole, ja muuttaa userIDStatus-kentän arvon todeksi. Paluuarvo on tosi, mikäli sijoitus onnistui. Tätä metodia on tarkoitus käyttää vain kahdessa tilanteessa: silloin, kun ServerConnection saa viestitysolion käyttäjältä, ja silloin kun jokin palvelimen komponentti lähettää viestin tietylle asiakkaalle.

getUserID -metodi antaa userID-kentän arvon.

ServerObject -luokka perii *CommunicationObject*-luokan, ja ylikirjoittaa replyClass-kentän siten, että vastauksen tyyppi voi olla vain *ClientObject*.

ClientObject -luokka perii *CommunicationObject*-luokan, ja ylikirjoittaa replyClass-kentän siten, että vastauksen tyyppi voi olla vain *ServerObject*.

5.2 ClientObject



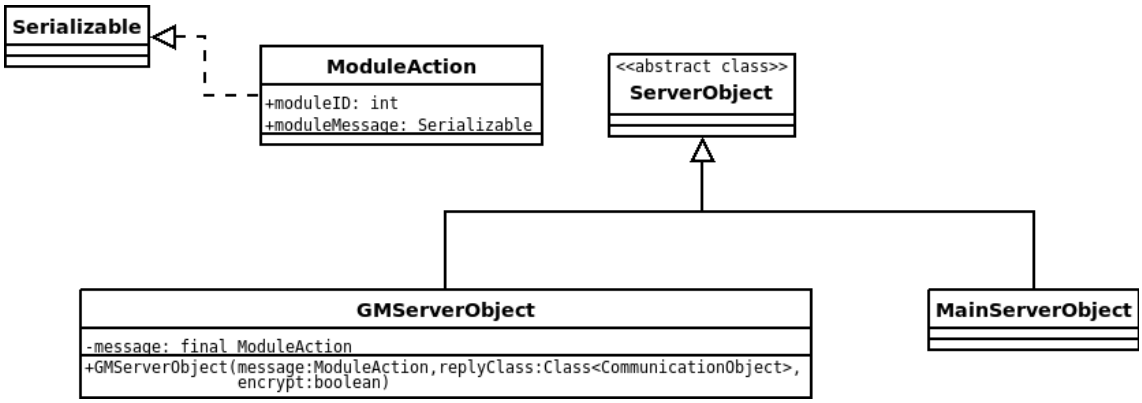
Kuva 7: ClientObject-luokat

ClientObject-luokan aliluokat ovat tyypin perusteella yksilöity jollekin asiakasohjelman komponentille. Kuvassa 7 näkyy Florida-asiakasohjelman käyttämien viestietysolioiden hierarkia.

GMClientObject -luokan ilmentymät on asiakasohjelman pelimoduulimanagerin käyttämiä viestitysolioita. Yliluokan *message*-kentän tyyppi on korvattu *ModuleAction*-luokalla, joka kapseloi moduulin käyttämän tiedon yksilöimällä sen.

MainClientObject on ClientMainin käyttämä viestitysluokka.

ClientGUIObject on *ClientGUIObjectManager*-luokan käyttämä viestitysluokka. Tämäkin kapseloi viestin *ModuleAction*-olioon.



Kuva 8: ServerObject-luokat

5.3 ServerObject

ServerObject-luokan aliluokat ovat *ClientObject*in tapaan yksilöity tyypin perusteella jollekin asiakasohjelman komponentille. Kuvassa 8 näkyy Florida-asiakasohjelman käyttämien viestietysolioiden hierarkia.

GMServerObject -luokan ilmentymät on palvelimen pelimoduulimanagerin käyttämiä viestitysolioita. Yliluokan *message*-kentän tyyppi on korvattu *ModuleAction*-luokalla, joka kapseloi moduulin käyttämän tiedon yksilöimällä sen.

MainServerObject on ServerMainin käyttämä viestitysluokka.

5.4 ServerDatabaseOperator

Tietokannan ja pelimoduulin palvelinkomponentin tiedonvälitykseen tarkoitettu olio.

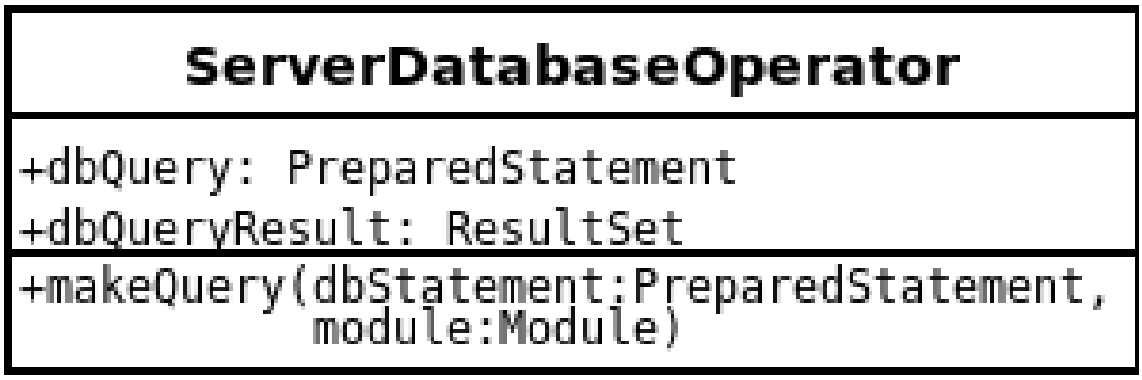
databaseQuery: PreparedStatement
Tehtävä tietokantaoperaatio SQL92-standardin mukaisesti. Operaation oikeellisuus on täysin palvelua pyytävän komponentin harteilla.

queryResult: ResultSet
Tietokantaoperaation tulos. ResultSet-olio palautetaan sellaisenaan palvelua pyytäneelle komponentille.

makeQuery(PreparedStatement dbQuery)
Metodi suorittaa tietokantaoperaation parametrina välitetyllä kyselyllä. Metodi palauttaa tietokantaoperaation tuloksen ResultSet muodossa.

makeQuery(PreparedStatement databaseQuery, Module module)
Metodi suorittaa tietokantaoperaation parametrina välitetyllä kyselyllä. Parametrinna an-

nettu moduli rajaa tietokannan käytön kyseiselle skeemalle (Schema). Metodi palauttaa tietokantaoperaation tuloksen ResultSet muodossa.



Kuva 9: ServerDatabaseOperator-rajapinnat

6 Luokat

6.1 Palvelin

ServerConnection

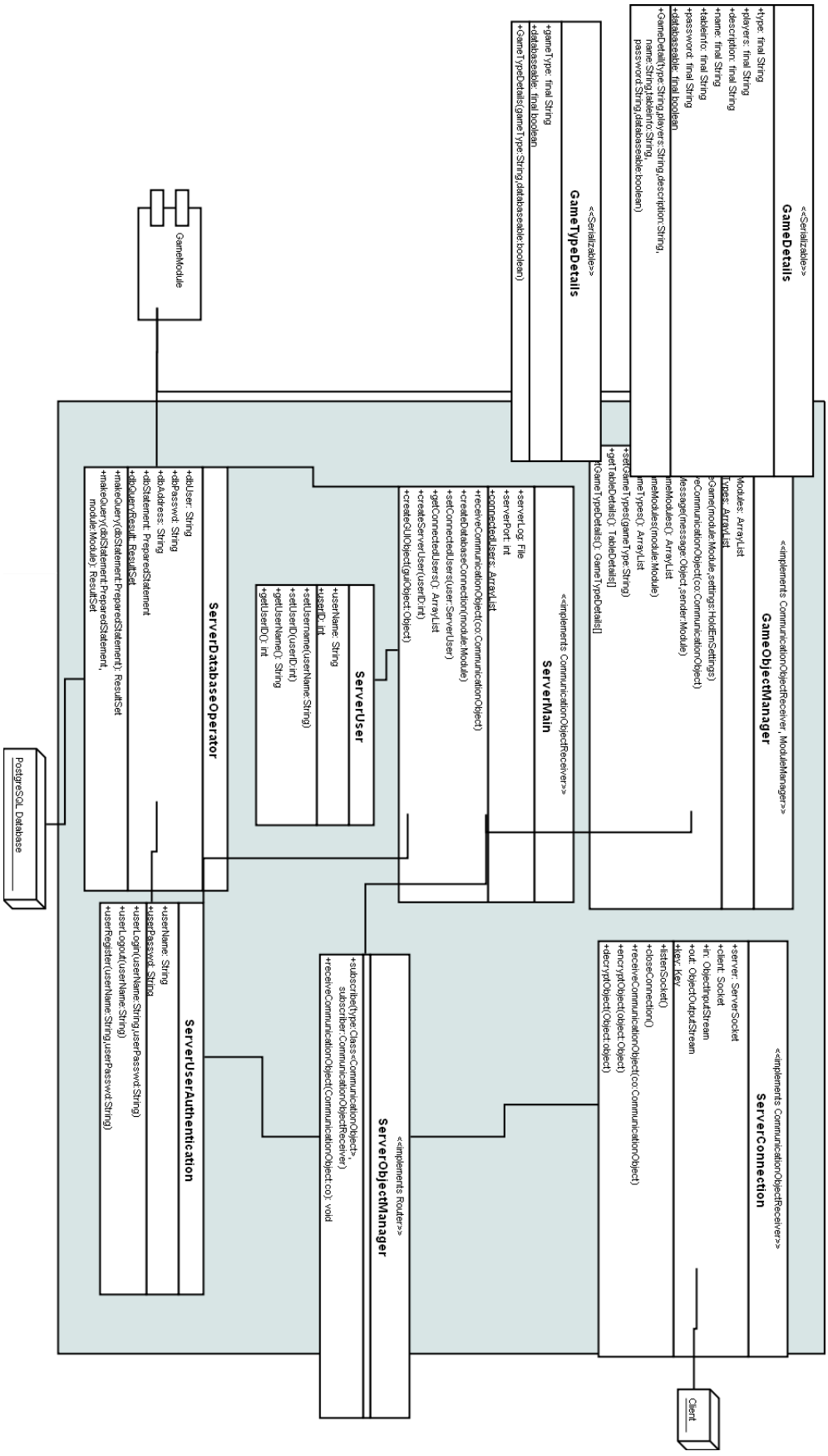
Kuvaus:
Luokka toteuttaa CommunicationObjectReceiver rajapinnan. Luokka ServerConnection mahdollistaa palvelimen ja asiakasohjelman välisen kommunikoinnin. Palvelin kuuntelee sisäänpäin tulevia yhteydenottoja määritellystä portista ja luo uusia yhteyksiä asiaskasohjelmiin Socket-olioiden avulla. Luokka toteuttaa myös tietoliikenteen salauksen. EncryptObject() ja decryptObject() hoitavat olioiden salaamisen ja salauksen purkamisen ennen kuin tieto reititetään ObjectManagerille. Luokka perii Thread-luokan, jotta sen ilmentymiä voidaa ajaa säikeinä.

Muuttujat:
`ServerSocket server`
server kuuntelee sisäänpäintulevia yhteydenottoja.

`Socket client`
client oliota käytetään palvelimen ja asiaskasohjelman väliseen TCP/IP datayhteyteen.

`ObjectInputStream out`
Ulospäin lähtevä data.

`ObjectOutputStream in`
Sisäänpäin tuleva data.



Kuva 10: Palvelimen luokkakaavio

`Key key`
Salauksen purkamiseen tarvittava avain.

Metodit:
`listenSocket()`
Metodi kuuntelee määriteltyä porttia.

`closeConnection()`
Metodi sulkee yhteyden määrättyyn asiakasohjelmaan.

`receiveCommunicationObject(CommunicationObject co)`
Otetaan vastaan parametrina saatu CommunicationObject-olio.

`encryptObject(Object object)`
Metodi salaa olion jonka jälkeen se on valmis lähetettäväksi verkon yli.

`decryptObject(Object object)`
Metodi purkaa olion salauksen jonka jälkeen se on valmis ohjelmiston käsiteltäväksi.

ServerObjectManager

Kuvaus:
Luokka toteuttaa Router rajapinnan. Luokka vastaa olioiden reitityksestä.

Muuttujat:
`private Map <Class<ServerObject>, ServerObjectReceiver> routingMap`
Tietorakenteessa on määritelty vastaanottaja kullekin edelleenlähetettävälle tietotyypille.

Metodit:
`subscribe(Class<CommunicationObject> type, CommunicationObjectReceive`
`subscriber )`
Pyytää ServerObjectManager-oliota lähettämään type-tyyppiset CommunicationObject-oliot subscriber-oliolle.

GameObjectManager

Kuvaus:
Luokka toteuttaa CommunicationObjectReceiver ja ServerGMManager -rajapinnat. Välittää peliobjektin oikealle pelimoduulille.

Muuttujat:
`Arraylist<String> gameModules`

Kaikki palvelimeen liitetyt pelimoduulit.

`Arraylist<String> gameTypes`
Kaikki palvelimen pelityypit. Jokaisella pelimoduulilla on joku tyyppi. Esimerkiksi Texas Hold'emin tyyppi on 'Poker'. Useampi pelimoduuli voi olla samaa tyyppiä.

Metodit:
`createGame(String module, HoldEmSettings settings)`
Luo uuden ilmentymän parametrina annetusta pelistä (module). Asetusmuuttuja (settings) määrittelee peliin liittyvät säännöt. Texas Hold'emin tapauksessa luodaan uusi pelipöytä annetuilla parametreilla.

`receiveCommunicationObject(CommunicationObject co)`
Metodi CommunicationObject-olion siirtämiselle.

`getGameModules()`
Palauttaa ArrayList-muodossa kutsujalle kaikki palvelimeen liitetyt pelimoduulit. Tiedot saadaaan gameModules ArrayList:istä.

`setGameModules(String module)`
Metodi saa parametrina Module-tyyppisen olion jonka se lisää gameModules ArrayListiin. Metodia ajetaan palvelimen käynnistyksen yhteydessä sekä kun uusi pelimoduuli on liitetty järjestelmään.

`getGameTypes()`
Palauttaa ArrayList-muodossa kutsujalle kaikki palvelin eri pelityypit. Tiedot saadaan gameTypes ArrayList:istä.

`setGameTypes(String gameType)`
Kun palvelin havaitsee uuden pelimoduulin jota ei vielä löydy gameTypes ArrayList:istä, lisätään ko. metodilla tyyppi pelien tyyppilistaukseen.

`sendGameModuleObject(Serialializable message, int user, Module sender, encrypt boolean)`
Toteuttaa ServerGMManager rajapinnan. Metodi lähettää viestitysolion.

`sendGUIObject(Serialializable message, int user, Module sender, boolean encrypt)`
Toteuttaa ServerGMManager rajapinnan. Metodi lähettää GUI-olion.

`checkForNewModules()`
Metodi tarkistaa onko palvelimeen liitetty uusia pelimoduleita. Tarkistus tehdään käynnistyksen yhteydessä sekä superuserin kirjautuessa järjestelmään. Mikäli uusi moduuli löydetään lisätään se järjestelmään kutsumalla setGameModules()-metodia.

ServerMain

Kuvaus:
Luokka palvelimen käynnistämiseen ja ylläpitämiseen. Luokka toteuttaa CommunicationObjectReceiver -rajapinnan.

Muuttujat:
`File serverLog`
Palvelimen lokitiedosto.

`int serverPort`
Palvelimen käyttämä portti.

`ArrayList<ServerUser> connectedUsers`
Kaikki käyttäjät jotka ovat yhteysdessä palvelimeen.

Metodit:
`receiveCommunicationObject(CommunicationObject co)`
Vastaanottaa CommunicationObject -olion. Jos CommunicationObject sisältää jonkin toiminnon, ServerMain toimii sen mukaan.

`createDatabaseConnection(String module)`
Metodi luo tietokantayhteyden kutsujalle. Kutsuja on joko GameObjectManager tai ServerUserAuthentication. Metodi palauttaa ServerDatabaseOperator-tyyppisen olion kutsujalle jonka avulla metodia kutsunut moduuli voi suorittaa omia tietokantaoperaatioita.

`setConnectedUsers(ServerUser user)`
Metodi saa parametrina ServerUser-olion jonka se lisää connectedUsers ArrayList:iin.

`getConnectedUsers()`
Metodi metodi palauttaa connectedUsers ArrayListin kutsujalle.

`createServerUser(int userID)`
Metodi luo ServerUser-tyyppisen olion annetulla userID:llä ja palauttaa ko. ServerUser-olion.

ServerUserAuthentication

Kuvaus:
Jokaisella palvelimeen yhteydessä olevalla käyttäjällä tulee olla voimassa oleva käyttäjätunnus. Luokalla tarkistetaan täsmäävätkö käyttäjätunnus ja salasana, sekä rekisteröidään uusia käyttäjätunnuksia.

Muuttujat:
`String userName`
Syötteenä saatu käyttäjätunnus.

`String userPasswd`
Syötteenä saatu salasana.

Metodit:
`userLogin()`
Metodi sisäänkirjautumiseen. Tarkistaa täsmäävätkö käyttäjän syöttämät käyttäjänimi ja salasana.

`userLogout()`
Metodi uloskirjautumiseen.

`userRegister()`
Uuden tunnuksen rekisteröintiin tarkoitettu metodi. Syötteenä saaduilla tiedoilla yritetään luoda uusi tunnus. Mikäli käyttäjänimeä ei vielä ole käytössä onnistuu rekisteröinti.

ServerDataBaseOperator

Kuvaus:
Toimii tietokantarajapintana PostgreSQL-tietokannan ja loppujärjestelmän välillä. Ainoastaan tällä luokalla on oikeudet tehdä tietokantakyselyjä.

Muuttujat:
`String dbUser`
Tietokannan käyttäjänimi.

`String dbPasswd`
Tietokannan salasana.

`String dbAddress`
Tietokannan osoite.

`PreparedStatement dbQuery`
Tehtävä kysely.

`ResultSet dbQueryResult`
Kyselyn tulokset.

Metodit:
`makeQuery(PreparedStatement dbQuery)`
Metodi suorittaa tietokantakyselyn parametrina annettulla PreparedStatement-lauseella.

Metodi palauttaa arvonaan kyselyn tuloksen ResultSet-muodossa.

`makeQuery(PreparedStatement dbQuery, Module module)`
Metodi suorittaa tietokantakyselyn parametrina annettulla PreparedStatement-lauseella. Muuttuja module rajaa kyselyn parametrina annetun moduulin omaan tietokanta skeemaan. Metodi palauttaa arvonaan kyselyn tuloksen ResultSet-muodossa.

ServerUser

Kuvaus:
Luokasta luodaan ServerUser-ilmentymiä. Jokaisesta palvelimelle onnistuneesti kirjautuneesta käyttäjästä luodaan ServerUser-olio.

Muuttujat:
`String userName`
Käyttäjän käyttäjänimi.

`int userID`
Käyttäjän yksilöivä ID.

Metodit:
`setUsername(String userName)`
Metodi asettaa userName muuttujan parametrina saaduksi String-olioksi.

`setUserID(int userID)`
Metodi asettaa userID muuttujan parametrina saaduksi int:ksi.

`getUsername()`
Metodi palauttaa ServerUser-olion tämänhetkisen userName String-olion.

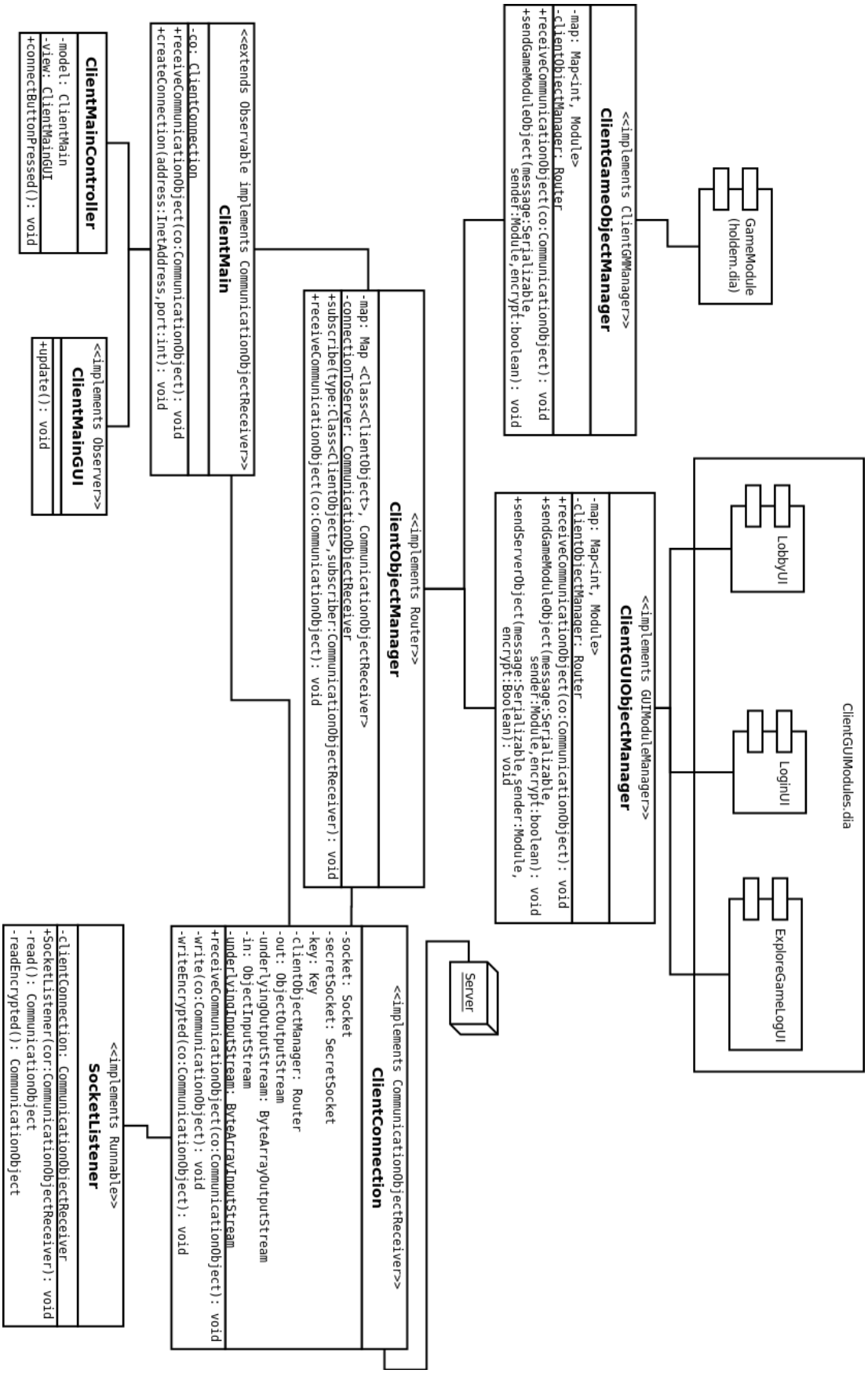
`getUserID()`
Metodi palauttaa ServerUser-olion tämänhetkisen userID int:in.

6.2 Asiakasohjelma

ClientMain

ClientMain extends Observable *Kuvaus:*
Tämä luokka on asiakasohjelman pääluokka, jossa toteutetaan Main-metodi. Luokka on myös vastuussa muiden pääkomponenttien kuten ClientConnectionin, ClientObjectManagerin, ClientGameObjectManagerin ja ClientGUIObjectManagerin alustamisesta ja niiden välisten yhteyksien luomisesta.

Muuttujat:



Kuva 11: Asiakasohjelman luokkakaavio

`ClientConnection co`
ClientConnection-olio, joka on yhteys asiakasohjelmalta palvelimeen.

Metodit:
`public void receiveCommunicationObject(CommunicationObject co)`
Ottaa vastaan CommunicationObject-olion ja käsittelee sen sisältämän viestin.

`private void createConnection(InetAddress address, int port) throws Exception`
Muodostaa ClientConnection-olion yhteyden annetuilla parametreilla.

ClientMainController *Kuvaus:*
Tämä luokka toteuttaa tapahtumienkäsittelyn ClientMainGUI-oliossa tapahtuville tapahtumille.

Muuttujat:
`private static ClientMain model`
Viite ClientMain-olioon, jotta tapahtumienkuuntelija voi käyttää sen metodeja.

`private static ClientGUI view`
Viite ClientGUI-olioon, jonka tapahtumiin reagoidaan.

Metodit:

ClientMainGUI implements observer *Kuvaus:*
Tämä luokka toteuttaa graafiset komponentit Connect to server-ikkunalle.

Muuttujat:

Metodit:

`public void update()`
Päivittää graafisen näkymän.

ClientConnection impelments CommunicationObjectReceiver

Kuvaus:
Tämä luokka sisältää yhteyden palvelimeen. Sen kautta voidaan lähettää ja vastaanottaa CommunicationObject-olioita.

Muuttujat:
`private Socket socket`
Pistoke palvelinkoneeseen

`private SecretSocket secretSocket`
Salattu pistoke palvelinkoneeseen.

`private Key key`
Avain salattuun pistokkeeseen.

`private Router clientObjectManager`
Viite Router-rajapinnan toteuttavaan olioon, jonne tulleet CommunicationObject-oliot siirretään reititystä varten.

`private ObjectOutputStream oos`
Objektivirta, johon kirjoitetaan Serializable-rajapinnan toteuttavia olioita. Tämä virta alustetaan antamalla parametrinä ByteArrayOutputStream ja liitetään socket-olioon.

`private ObjectInputStream ois`
Objektivirta, josta otetaan vastaan Serializable-rajapinnan toteuttavia olioita. Tämä virta alustetaan antamalla parametrinä ByteArrayInputStream ja liitetään socket-olioon.

`private ObjectOutputStream oosEncrypted`
Objektivirta, johon kirjoitetaan Serializable-rajapinnan toteuttavia olioita. Tämä virta alustetaan antamalla parametrinä ByteArrayOutputStream ja liitetään secureSocket-olioon.

`private ObjectInputStream oisEncrypted`
Objektivirta, josta otetaan vastaan Serializable-rajapinnan toteuttavia olioita. Tämä virta alustetaan antamalla parametrinä ByteArrayInputStream ja liitetään secureSocket-olioon.

Metodit:
`void public receiveCommunicationObject(CommunicationObject co)`
Ottaa vastaan CommunicationObject-olioita ja reitittää sen joko salaamattomaan tai salattuun pistokkeeseen tai clientObjectManagerille riippuen CommunicationObject-olion tyypistä.

`void private write(CommunicationObject co)`
Kirjoittaa annetun CommunicationObject-olion oos-olioon.

`void private writeEncrypted(CommunicationObject co)`
Kirjoittaa annetun CommunicationObject-olion oosEncrypted-olioon.

`CommunicationObject private read()`
Lukee oos-virtaa.

`CommunicationObject private readEncrypted()`
Lukee oosEncrypted-virtaa.

ClientObjectManager implements Router *Kuvaus:*
Asiakasohjelman CommunicationObject reitittäjä. Reitittää CommunicationObject-oliot niiden tyyppien mukaan eteenpäin.

Muuttujat:
`private Map <Class<ClientObject>, CommunicationObjectReceiver`
Tietorakenteessa on määritelty vastaanottaja kullekin edelleenlähetettävälle tietotyypille.

`private CommunicationObjectReceiver connectionToServer`
Palvelinyhteys

Metodit:
`public void subscribe(Class<ClientObject> type, CommunicationObjectRe`
`subscriber)`
Pyydetään ClientObjectManager-oliota lähettämään saamansa *type*-tyyppiset ClientObject-oliot *subscriber*-oliolle.

`public void receiveCommunicationObject(CommunicationObject co)`
Otetaan vastaan CommunicationObject-olio ja reititetään sen tyypin mukaan (Map-tietorakenteessa) oikealle komponentille.

ClientGameObjectManager implements ClientGMManager, CommunicationObjectReceiver *Kuvaus:*
ClientGameObjectManager ottaa vastaan ClientGameObject-olioita ja reitittää ne oikeille

vastaanottajamoduuleille. Luokka on myös vastuussa CommunicationObject-olioiden luomisesta.
Muuttujat:
`Map<int, Module>`
Viitetaulukko, joka yhdistää moduulin tunnuksen viitteeseen kyseiseen olioon.

`Router clientObjectManager`
CommunicationObject-olioiden reititin, jonne moduuleilta lähtevät oliot lähetetään.

Metodit:
`public void receiveMessage(Object message, Module sender)`
Ottaa vastaan olion ja paketoi sen CommunicationObject-olioksi, jonka jälkeen laittaa sen eteenpäin.

`public void receiveCommunicationObject(CommunicationObject co)`
Ottaa vastaan CommunicationObject-olion, jonka jälkeen selvittää mille moduulille se kuuluu ja laittaa sen eteenpäin.

ClientGUIObjectManager implements GUIModuleManager, CommunicationObjectReceiver *Kuvaus:*
ClientGUIObjectManager ottaa vastaan ClientGUIObject-olioita ja reitittää ne oikeille vastaanottajamoduuleille. Luokka on myös vastuussa CommunicationObject-olioiden luomisesta.
Muuttujat:
`Map<int, Module>`
Viitetaulukko, joka yhdistää moduulin tunnuksen viitteeseen kyseiseen olioon.

`Router clientObjectManager`
CommunicationObject-olioiden reititin, jonne moduuleilta lähtevät oliot lähetetään.

Metodit:
`public void receiveMessage(Object message, Module sender)`
Ottaa vastaan olion ja paketoi sen CommunicationObject-olioksi, jonka jälkeen laittaa sen eteenpäin.

`public void receiveCommunicationObject(CommunicationObject co)`
Ottaa vastaan CommunicationObject-olion, jonka jälkeen selvittää mille moduulille se kuuluu ja laittaa sen eteenpäin

6.3 Asiakasohjelman graafiset käyttöliittymät palvelimelta

Nämä kuvassa 12 näkyvät käyttöliittymät ladataan palvelimelta niitä käytettäessä.

6.4 Pelimoduuli

Game

Kuvaus:
Rajapintaluokka Game määrittelee kaikkia järjestelmään liitettäviä pelimoduuleita koskevat muuttujat ja metodit. Game on GameServerModulen aliluokka.

Muuttujat:
`private List<User> players`
Lista pelipöydässä olevista pelaajista.

`private String name`
Pelin nimi, esim. Texas Hold'em.

`private String type`
Pelityyppi, esim. pokeri.

`private String tableName`
Pelipöydän nimi

`private User createdBy`
Pelipöydän luoja

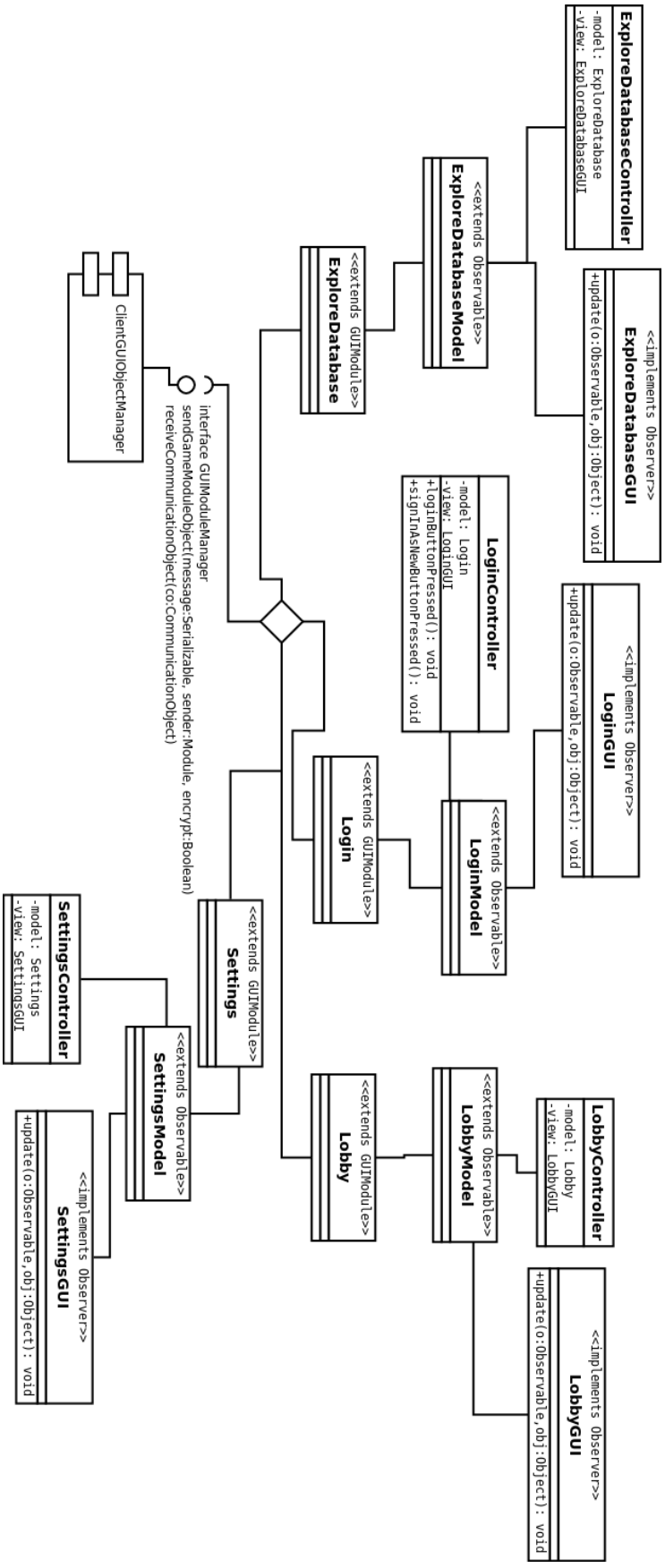
`private databaseOperator dbOperator`
Tietokantaoperaatioiden tekemiseen käytettävä olio

Metodit:
`List<User> getPlayers()`
Palautetaan pelaajalista.

`String getName()`
Palautetaan pelin nimi.

`String getType()`
Palautetaan pelityyppi.

`GameGUI getGUI()`
Palautetaan pelin graafisen käyttöliittymän toteuttava GameGUI-olio.



Kuva 12: Asiakasohjelman käyttöliittymät(Palvelimelta)

`GameEventHandler getGameEventHandler()`
Palautetaan graafisen käyttöliittymän tapahtumanhallinnan suorittava GameEventHandler-olio.

`List<Method> getMethods()`
Palautetaan lista pelimoduulin toteuttamista metodeista.

`String getSQLCreateTable()`
Palautetaan SQL-kielen CREATE TABLE-lause. Luotuun tauluun tallennetaan pelitietoja.

`String getSQLInsert()`
Palautetaan SQL-kielen INSERT-lause, jolla tallennetaan pelitietoja.

`String getTableName()`
Palautetaan pelipöydän nimi.

`setTableName(String tn)`
Asetetetaan pelipöydän nimi parametrina saaduksi.

`String getCreatedBy()`
Palautetaan pelipöydän luoja.

`setCreatedBy(int id, String name`
Asetetaan pelipöydän luoja saatujen parametrien perusteella muodostetuksi User-olioksi.

`GameClient getClient()`
Palautetaan GameClient-olio, jonka avulla määritellään pelin graafinen käyttöliittymä, sen tapahtumakuuntelu ja tapahtumien välittäminen palvelinpuolen pelimoottorilta ja -moottorille.

`private parse(GameServerModuleObject gsmo)`
Käsitellään vastaanotettu GameServerModuleObject, jonka sisältönä on asiakkaalta tullut pelitapahtumatieto tai chatviesti. Chatviesti välitetään muille pelipöytään kiinnittyneille asiakkaille. Pelitapahtumatiedon avulla päivitetään pelin tilatieto, joka sekin sitten välitetään pelipöytään kiinnittyneille asiakkaille.

GameClient

Kuvaus:
Luokka GameClient perii GUIModulen. GameClient apuluokkineen määrittelee pelin graafisen käyttöliittymän, sen tapahtumakuuntelun ja tapahtumien välittämisen palvelinpuolen pelimoottorilta ja -moottorille.

Muuttujat:
`private GameClientController controller`
Pelin asiakasohjelman tapahtumakäsittelijä

`private GameClientModel model`
Pelin asiakasohjelman tietosisältö

GameClientController

Kuvaus:
Luokka GameClientController on pelin asiakasohjelman tapahtumakäsittelijä. Sen rooli ja toiminta perustuvat MVC-suunnittelumalliin.

Muuttujat:
`private GameClientModel model`
Pelin asiakasohjelman tietosisältö

`private GameClientGUI view`
Pelin asiakasohjelman käyttöliittymä

`protected ArrayList<Class>`
Lista pelin asiakasohjelman tarvitsemista luokista. Nämä luokat saattavat olla tarpeen esim. botin ajamisessa

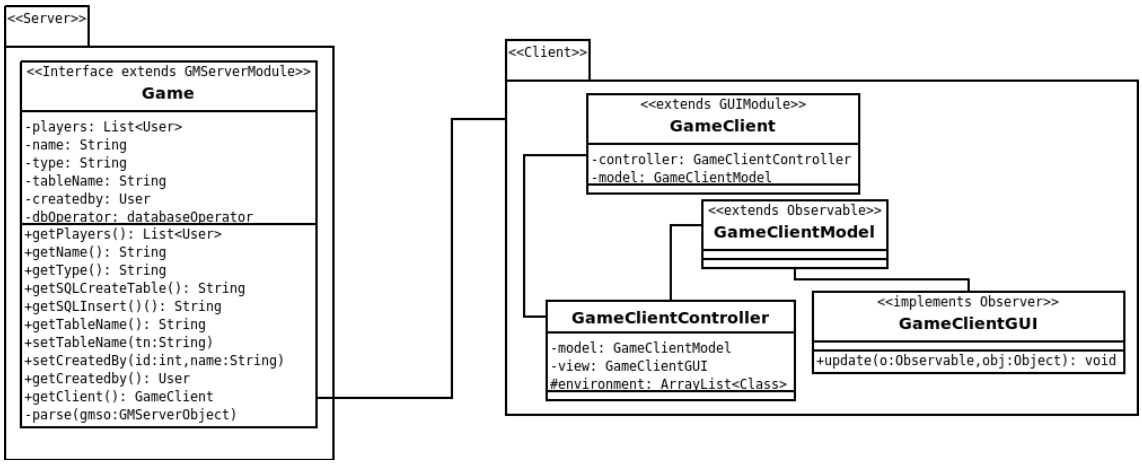
GameClientModel

Kuvaus:
Luokka GameClientModel perii java.util.Observable-luokan. GameClientModel määrittelee pelin asiakasohjelman tietosisällön. Sen rooli ja toiminta perustuvat MVC-suunnittelumalliin.

GameClientGUI

Kuvaus:
Luokka GameClientGUI toteuttaa java.util.Observer-rajapintaluokan. GameClientGUI määrittelee pelin asiakasohjelman käyttöliittymän. Käyttöliittymää päivitetään tietosisällön muuttuessa GameClientModel-oliossa.

Metodit:
`void update(Observable o, Object obj`
Observer-rajapinnan määrittelemä metodi, jota kutsutaa, kun tarkkailtavassa kohteessa on ollut muutoksia. Parametreina saadaan tarkkailtava kohde ja argumentti.



Kuva 13: Pelimoduuli

TransmittableAction

Kuvaus:
Luokka TransmittableAction määrittelee oliot, joiden välityksellä pelimoduulin asiakaspuoli ja pelimoottori kommunikoivat.

TexasHoldEm

Kuvaus:
Luokka TexasHoldEm on peli, joka toteuttaa rajapintaluokan Game.

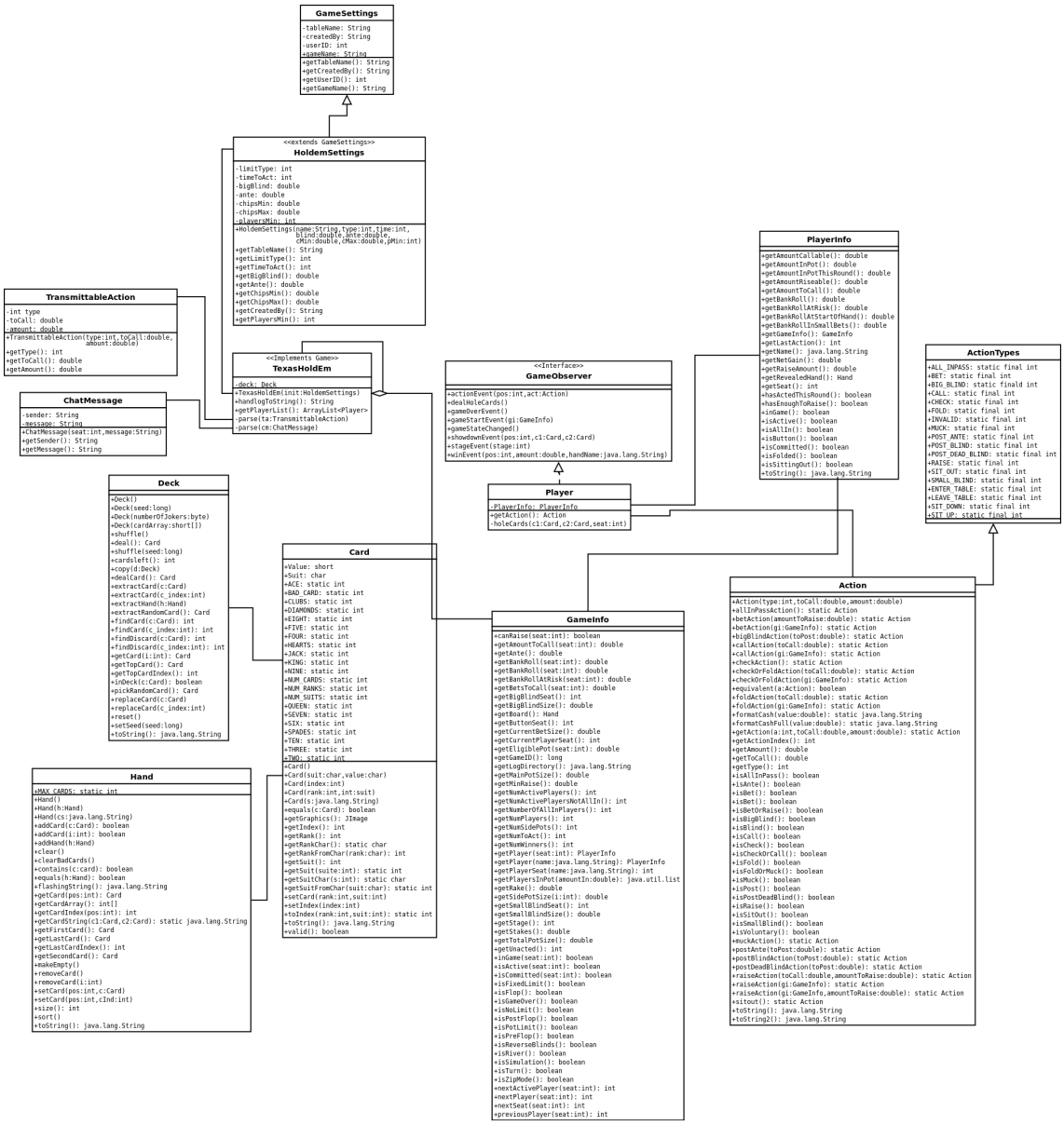
Muuttujat:
`Deck deck`
Korttipakka, joka on yksi Deck-olio.

Metodit:
`TexasHoldEm(HoldemSettings init)`
Luodaan ilmentymä parametrina saaduin asetuksin. Yksi TexasHoldEm-ilmentymä vastaa yhtä pelipöytää.

`String dbInsertToString()`
Palautetaan String-olio pelitiedon tietokantaan viemistä varten.

`String handlogToString()`
Palautetaan String-olio käsilokitiedostoon viemistä varten.

`Arraylist<Players> getPlayers()`
Palautetaan lista pelipöydässä olevista pelaajista ArrayList-oliona, jossa alkioina on Player-olioita.



Kuva 14: Texas Hold’em -luokat

`private parse(ChatMessage cm)`
Käsitellään vastaanotettu chatviesti.

`private parse(TransmittableAction ta)`
Käsitellään vastaanotettu pelitapahtumatieto.

GameSettings

Kuvaus:
Luokka GameSettings määrittelee pelipöydän luomisessa käytettävän asetusolion.

Muuttujat:
`private String tableName`
Pelipöydän nimi

`private String createdBy`
Pelipöydän luoja

`private int userID`
Pelipöydän luojan tunnus

`private String gameName`
Pelin nimi, esim. Texas Hold em

Metodit:
`String getTableName()`
Palautetaan pelipöydän nimi

`String getCreatedBy()`
Palautetaan pelipöydän luojan nimi

`String getUserID()`
Palautetaan pelipöydän luojan tunnus

`String getGameName()`
Palautetaan pelipöydässä pelattavan pelin nimi

HoldEmSettings

Kuvaus:
Luokka HoldemSettings määrittelee pelipöydän asetusolion, jonka avulla luodaan pelipöytä,

jossa pelataan Texas Hold 'em -pokeria. HoldemSettings perii GameSettings-luokan.

Muuttujat:
`private int limitType`
Käytettävä panosrajoitustyyppi

`private int timeToAct`
Vuoron aikaraja

`private double bigBlind`
Ison sokkopanoksen koko

`private double ante`
Aloituspanoksen koko

`private double chipsMin`
Pöytään tuotava vähimmäispelimerkkimäärä

`private double chipsMax`
Pöytään tuotava enimmäispelimerkkimäärä

`private int playersMin`
Vähimmäispelaajamäärä, joka tarvitaan pelipöytään ennen kuin peli käynnistyy

Metodit:
`HoldemSettings(String name, int type, int time, double blind, double ante, double cMin, double cMax, int pMin)`
Luodaan HoldemSettings-olio saatujen parametrien avulla.

`String getTableName()`
Palautetaan pelipöydän nimi.

`int getLimitType()`
Palautetaan käytettävä panostusrajoitustyyppi.

`int getTimeToAct()`
Palautetaan vuoron aikaraja.

`double getBigBlind()`
Palautetaan ison sokkopanoksen koko.

`double getAnte()`

Palautetaan aloituspanoksen koko.

`double getChipsMin()`
Palautetaan pöytään tuotava vähimmäispelimerkkimäärä.

`double getChipsMax()`
Palautetaan pöytään tuotava enimmäispelimerkkimäärä.

Deck

Kuvaus:
Luokka Deck määrittelee korttipeleissä käytettävän korttipakan. Luokka on yhteensopiva Meerkat API:ssa määritellyn Deck-luokan kanssa.

Metodit:
`Deck()`
Luodaan 52 kortin sekoittamaton korttipakka.

`Deck(long seed)`
Luodaan 52 kortin saadulla siemenluvulla sekoitettu korttipakka.

`Deck(byte numberOfJokers)`
Luodaan korttipakka, jossa on 52 kortin lisäksi parametrina annettu määrä jokereita.

`Deck(short[] cardArray)`
Luodaan korttipakka parametrina saadun taulukon korteista.

`shuffle()`
Sekoitetaan pakka.

`shuffle(long seed)`
Sekoitetaan pakka parametrina saadulla siemenluvulla.

`Card deal()`
Jaetaan pakasta päällimmäinen kortti.

`Card dealCard()`
Jaetaan pakasta satunnainen kortti.

`int cardsLeft()`
Palautetaan pakassa jäljellä olevien korttien määrä.

`copy(Deck d)`
Kopioidaan parametrina saatu pakka kutsuvaan pakkaan.

`extractCard(Card c)`
Poistetaan parametrina saatu kortti pakasta.

`extractCard(int c_index)`
Poistetaan parametrina saadussa kohtaa sekoittamattonta pakkaa oleva kortti pakasta.

`extractHand(Hand h)`
Poistetaan parametrina saadun käden kortit pakasta.

`Card extractRandomCard()`
Poistetaan ja palautetaan satunnainen kortti pakasta.

`int findCard(Card c)`
Etsitään pakasta parametrina saatu kortti ja palautetaan sen sijainti pakassa.

`int findCard(int c_index)`
Etsitään parametrina saadussa kohdassa sekoittamattomassa pakkassa sijaitseva kortti pakasta ja palautetaan sen sijainti pakassa.

`int findDiscard(Card c)`
Etsitään parametrina saatu kortti, palautetaan sen sijainti pakassa ja poistetaan se.

`int findDiscard(int c_index)`
Etsitään parametrissa saadussa kohdassa sekoittamattomassa pakassa oleva kortti, palautetaan sen sijainti ja poistetaan se.

`Card getCard(int i)`
Palautetaan parametrina saadussa kohdassa pakkaa sijaitseva kortti.

`Card getTopCard()`
Palautetaan pakan päällimmäinen kortti. Korttia ei poisteta pakasta.

`boolean inDeck(Card c)`
Palautetaan tosi, jos parametrina saatu kortti on pakassa ja epätosi, jos kortti ei ole pakassa.

`Card pickRandomCard()`
Palautetaan pakasta satunnainen kortti poistamatta sitä pakastaö.

`void replaceCard(Card c)`
Pannaan parametrina saatu kortti takaisin pakkaan.

`void replaceCard(int c_index)`
Pannaan parametrina saadussa kohdassa sekoittamattomassa pakassa oleva kortti takaisin

pakkaan.

`void reset()`
Pannaan kaikki kortit takaisin pakkaan. Pakkaa ei järjestetä.

`void setSeed(long seed)`
Asetetaan pakan sekoittamiseen käytettävä siemenluku parametrina saaduksi, jos seed <> nolla. Jos seed = nolla, siemenluku pidetään ennallaan.

`String toString()`
Palautetaan pakan merkkijonoesitys.

Hand

Kuvaus:
Luokka Hand määrittelee korkeintaan seitsemästä kortista (Card) muodostuvan käden. Hand toteuttaa java.io.Serializable-rajapinnan.

Muuttujat:
`static int MAX_CARDS`
Korttien maksimimäärä.

Metodit:
`Hand()`
Luodaan tyhjä käsi.

`Hand(Hand h)`
Luodaan parametrina saadun käden kopio.

`Hand(String cs)`
Luodaan käsi parametrina saadun merkkijonoesityksen perusteella.

`boolean addCard(Card c)`
Lisätään parametrina saatu kortti käteen, jos kädessä on tilaa. Jos parametri on *null*, lisäysyritys jätetään huomiotta. Palautetaan tosi, jos kortti lisättiin ja epätosi, jos korttia ei lisätty.

`boolean addCard(int i)`
Lisätään parametrina saadussa kohdassa sekoittamattomassa pakassa oleva kortti käteen, jos kädessä on tilaa. Palautetaan tosi, jos lisäys onnistui ja epätosi, jos lisäys ei onnistunut.

`void addHand(Hand h)`
Lisätään parametrina saadun käden kortit käteen.

`void clear()`
Poistetaan kortit kädestä.

`void clearBadCards()`
Poistetaan kelvottomat (sic) kortit kädestä.

`boolean contains(Card c)`
Palautetaan tosi, jos kädessä on parametrina saatu kortti. Muuten palautetaan epätosi.

`boolean equals(Hand h)`
Palautetaan tosi, jos parametrina saatu käsi sisältää samat kortit kuin kutsuja. Muuten palautetaan epätosi.

`String flashingString()`
Palautetaan käden merkkijonoesitys käden näyttämistä varten. (Kortit näytetään vastustajan lukemiseksi. Kädestä ei luovuta.)

`Card getCard(int pos)`
Palautetaan kortti, joka on parametrina saadussa kohdassa kättä. Kortit on numeroitu 1 – 7.

`int[] getCardArray()`
Palautetaan taulukko, jossa ensimmäisenä alkiona on kädessä olevien korttien määrä. Sitä seuraavissa alkioissa on korttien sijainnit sekoittamattomassa pakassa.

`int getCardIndex(int pos)`
Palautetaan kädessä parametrina saadussa olevan kortin sijainti sekoittamattomassa pakassa.

`String getCardString(Card c1, Card c2,)`
Palautetaan parametreina saatujen korttien merkkijonoesitys.

`Card getFirstCard()`
Palautetaan käden ensimmäinen kortti eli ensimmäinen piilokortti.

`Card getLastCard()`
Palautetaan käden viimeinen kortti.

`int getLastCardIndex()`
Palautetaan käden viimeisen kortin sijainti sekoittamattomassa pakassa.

`Card getSecondCard()`
Palautetaan käden toinen kortti eli toinen piilokortti.

`void makeEmpty()`
Poistetaan kaikki kortit kädestä.

`void removeCard()`
Poistetaan käden viimeinen kortti.

`void removeCard(int i)`
Poistetaan kädestä parametrina saadussa kohdassa oleva kortti.

`void setCard(int pos, Card c)`
Lisätään parametrina saatuna kortti parametrinan saatuun paikkaan kädessä.

`void setCard(int pos, int cInd)`
Lisätään sekoittamattomassa pakassa parametrina *pos* saadussa kohdassa oleva kortti parametrina *c* saatuun paikkaan kädessä.

`int size()`
Palautetaan kädessä olevien korttien määrä.

`void sort()`
Kuplajärjestetetään käsi suurimmasta pienimpään sekoittamattomassa pakassa sijainnin perusteella.

`String toString()`
Palautetaan käden merkkijonoesitys.

GameInfo

Kuvaus:
Luokka GameInfo on rajapintaluokka, joka määrittelee kaiken käynnissä olevaan jakoon liittyvän julkisen tiedon.

Metodit:
`boolean canRaise(int seat)`
Palautetaan tosi, jos parametrina saadulla paikalla istuva pelaaja voi korottaa.Palautetaan pelimerkkimäärä, jonka parametrina saadulla paikalla istuvalta pelaajalta vaaditaan katsomiseen.

`double getAmountToCall(int seat`
Palautetaan pelimerkkimäärä, jonka parametrina saadulla paikalla istuvalta pelaajalta vaaditaan katsomiseen.

`double getAnte()`
Palautetaan aloituspanoksen koko.

`double getBankRoll(int seat)`
Palautetaan pelimerkkimäärä, joka parametrina saadulla paikalla istuvalla pelaajalla on

käytettävissään.

`double getBankRollAtRisk(int seat)`
Palautetaan pelimerkkimäärä, jonka verran parametrina saadulla paikalla istuvalla pelaajalla on korkeintaan mahdollisuus menettää jaossa. Määrä on suurin mukana oleva merkkimäärä, jos pelaajalla on enemmän pelimerkkejä tai pelaajan koko merkkimäärä, suurin mukana oleva merkkimäärä on pelaajan merkkimäärää suurempi.

`double getBetsToCall(int seat)`
Palautetaan katsomiseen vaadittava pelimerkkimäärä panoksina (jos panoksen koko on rajoitettu).

`int getBigBlindSeat()`
Palautetaan ison sokkopanoksen maksaneen pelaajan paikka.

`double getBigBlindSize()`
Palautetaan ison sokkopanoksen koko.

`Hand getBoard()`
Palautetaan pöytäkortit Hand-oliona.

`int getButtonSeat()`
Palautetaan jakajan paikka.

`double getCurrentBetSize()`
Palautetaan panoksen koko (jos rajoitettu).
`int getCurrentPlayerSeat()`
Palautetaan vuorossa olevan pelaajan paikka.

`double getEligiblePot(int seat)`
Palautetaan pelimerkkimäärä, jonka parametrina annetulla paikassa oleve pelaaja voi jaossa voittaa. Pelaaja voi voittaa merkkejä vain niistä poteista, joissa on hänen omia merkkejään mukana.

`long getGameID()`
Palautetaan jaon yksikäsitteinen tunniste.

`lava.lang.String getLogDirectory()`
Metodi toteutetaan tynkänä yhteensopivuuden vuoksi. Metodin kutsuminen aiheuttaa ajonaikaisen poikkeuksen, Unsupported Operation Exceptionin.

`double getMainPotSize()` Palautetaan pääpotin koko.

`double getMinRaise()`
Palautetaan pienimmän sallitun korotuksen koko.

`int getNumActivePlayers()`
Palautetaan vielä jaossa mukana olevien pelaajien määrä (osallistujat - korteistaan luopuneet pelaajat).

`int getNumActivePlayersNotAllIn()`
Palautetaan pelaajien määrä, jotka eivät ole jaossa all-in.

`int getNumberOfAllInPlayers()`
Palautetaan pelaajien määrä, jotka ovat jaossa all-in.

`int getNumPlayers()`
Palautetaan jakoon osallistuneiden pelaajien määrä.

`int getNumRaises()`
Palautetaan jaossa tehtyjen panostusten ja korotusten yhteismäärä.
`int getNumSeats()`
Palautetaan peliin osallistujien määrälle asetettu yläraja.

`int getNumSidePots()`
Palautetaan sivupottien määrä.

`int getNumToAct()`
Palautetaan tällä panostuskierroksella vuoroon tulevien pelaajien määrä olettaen, ettei uusia korotuksia tehdä.

`int getNumWinners()`
Palautetaan pelaajien määrä, jotka voittivat jaossa.

`PlayerInfo getPlayer(int seat)`
Palautetaan parametrina saadulla paikalla olevan pelaajan tilatieto Playerinfo-oliona.

`PlayerInfo getPlayer(java.lang.String name)`
Palautetaan parametrina saadun nimisen pelaajan tilatieto Playerinfo-oliona.

`int getPlayerSeat(java.lang.String name)`
Palautetaan parametrina saadun nimisen pelaajan paikka.

`java.util.list getPlayersInPot(double amountIn)`
Palautetaan lista pelaajista, jotka voivat voittaa potin ja joilla on vähintään parametrina annetun määrän verran pelimerkkejä mukana potissa.

`double getRake()`
Palautetaan pelimerkkimäärä, jonka pelinjärjestäjä on ottanut itselleen palkkioksi potista.

`double getSidePotSize(int i)`
Palautetaan parametrina saadun sivupotin koko. Parametri i on väliltä 0..n-1, n on sivupottien määrä.

`int getSmallBlindSeat()`
Palautetaan paikka, jolla oleva pelaaja on maksanut pienen sokkopanoksen. Palautettavan paikan valintaan vaikuttava pieni sokkopanos voi olla maksettu myös jakojen väliin jättämisen takia.

`double getSmallBlindSize()`
Palautetaan pienen sokkopanoksen koko.

`int getStage()`
Palautetaan meneillään oleva pelivaihe (Holdem.PREFLOP, Holdem.FLOP, Holdem.TURN, Holdem.RIVER).

`double getStakes()`
Palautetaan pelimerkkimäärä, joka pelaajan on täytynyt sijoittaa peliin pysyäkseen jaossa mukana.

`double getTotalPotSize()`
Palautetaan pääpotin ja sivupottien yhteenlaskettu koko.

`int getUnacted()`
Palautetaan niiden pelaajien määrä, jotka vielä tulevat tämän kierroksen aikana vuoroon.

`boolean inGame(int seat)`
Palautetaan tosi, jos parametrina annetulla paikalla oleva pelaaja on osallistunut jakoon, ts. ei ole jättänyt jakoa väliin (sit-out)

`boolean isActive(int seat)`
Palautetaan tosi, jos parametrinan annetulla paikalla olevan pelaaja on vielä mukana jaossa, ts. osallistunut jakoon eikä ole luopunut korteistaan. Muuten palautetaan epätosi.

`boolean isCommitted(int seat)`
Palautetaan tosi, jos parametrina annetulla paikalla oleva pelaaja on sijottanut menossa olevalla kierroksella pelimerkkejä. Muuten palautetaan epätosi.

`boolean isFixedLimit()`
Palautetaan tosi, jos pelimuoto on “fixed limit“ eli panoksen enimmäiskoko on kiinnitetty.

`boolean isFlop()`
Palautetaan tosi, jos menossa on

`boolean isGameOver()`

Palautetaan tosi, jos jako on päättynyt. Muuten palautetaan epätosi.

`boolean isNoLimit()`
Palautetaan tosi, jos pelimuoto on “no limit“ eli panoksen enimmäiskokoa ei ole rajoitettu.

`boolean isPostFlop()`
Palautetaan tosi, jos menossa ei ole ensimmäinen panostuskierros (pre-flop). Muuten palautetaan epätosi.

`boolean isPotLimit()`
Palautetaan tosi, jos pelimuoto on “pot limit“ eli suurin mahdollinen panostus tai korotus on yhtä suuri kuin potin koko.

`boolean isPreFlop()`
Palautetaan tosi, jos menossa on ensimmäinen panostuskierros (pre-flop.) Muuten palautetaan epätosi.

`boolean isReverseBlinds()`
Palautetaan tosi, jos kaksinpelissä jakajan paikalla oleva pelaaja maksaa pienen sokkopanoksen. Muuten palautetaan epätosi.

`boolean isRiver()`
Palautetaan tosi, jos menossa on neljäs panostuskierros (river).

`boolean isSimulation()`
Palautetaan aina epätosi.

`boolean isTurn()`
Palautetaan tosi, jos menossa on kolmas panostuskierros (turn). Muuten palautetaan epätosi.

`boolean isZipMode()`
Palautetaan aina epätosi.

`int nextActivePlayer(int seat)`
Palautetaan parametrina annetulla paikalla olevaa pelaajaa seuraan, jaossa vielä mukana olevan pelaajan paikka. Palautettava ei voi olla tyhjä tai korteistaan luopuneen pelaajan paikka.

`int nextPlayer(int seat)`
Palautetaan parametrina annetulla paikalla olevaa pelaajaa seuraavan pelaajan paikka. Palautettava ei voi olla tyhjä paikka.

`int nextSeat(int seat)`
Palautetaan parametrina saatua paikkaa seuraava paikka.

`int previousPlayer(int seat)`
Palautetaan parametrina annetulla paikalla olevaa pelaajaa edeltävän pelaajan paikka. Palautettava ei voi olla tyhjä paikka.

GameObserver

Kuvaus:
Luokka GameObserver on rajapintaluokka, joka määrittelee pokeripeliä seuraavalta oliolta vaadittavat ominaisuudet. Kaikki julkiset pelitapahtumat lähetetään rajapinnan toteuttajalle mahdollistaen pelin seuraamisen.

Metodit:
`actionEvent(int pos, Action act)`
Otetaan vastaan parametrina saadulla paikalla olevan pelaajan toiminto, joka saadaan toisena parametrina. Metodin voi kuormittaa saadakseen tiedot tietyn pelaajan kaikista toiminnoista.

`dealHoleCardsEvent()`
Käsikortit jaetaan.

`gameOverEvent()`
Jako on päättynyt.

`gameStartEvent(GameInfo gi)`
Jako alkaa. Parametrina saadaan GameInfo-olio, joka määrittelee jaon tilatiedot.

`gameStateChanged()`
Jaon tilatieto on muuttunut. Metodia kutsutaan, kun jonkin pelaajan toiminto on käsitelty kokonaan.

`showdownEvent(int pos, Card c1, Card cd)`
Parametrina saadulla paikalla oleva pelaaja on näyttänyt toisena ja kolmantena parametrina saadut kortit.

`stageEvent(int stage)`
Uusi panostuskierros on alkanut.

`winEvent(int pos, double amount, String handName)`
Ensimmäisenä parametrina saadulla paikalla oleva pelaaja on voittanut toisena parametrina saadun pelimerkkimäärän kolmantena parametrina saadun merkkijonon esittämällä kädellä.

Player

Kuvaus:
Luokka Player on GameObserverin alirajapintaluokka. Siinä on määritelty pokerin pelaamiseen vaadittavat metodit.

Metodit:
`Action getAction()`
Pelaajalta pyydetään toimintoa. Metodia kutsutaan, kun on pelaajan vuoro.

`holeCards(Card c1, Card c2, int seat`
Otetaan vastaan parametreina saadut käsikortit. Kolmas parametri on pelaajan paikka.

`init(Preferences prefs)`
Metodi toteutetaan tynkänä yhteensopivuuden vuoksi. Metodin kutsuminen aiheuttaa ajonaikaisen poikkeuksen, Unsupported Operation Exceptionin.

PlayerInfo

Kuvaus:
Luokka PlayerInfo on rajapintaluokka, joka määrittelee pelaajan julkisen tilatiedon meneillään olevassa jaossa.

Metodit:
`double getAmountCallable()`
Palautetaan pelimerkkimäärä, jonka pelaaja voi maksaa tehdystä panostuksesta.

`double getAmountInPot()`
Palautetaan pelimerkkimäärä, joka pelaajalla on potissa.

`double getAmountInPotThisRound()`
Palautetaan pelimerkkimäärä, jonka pelaaja on tällä kierroksella pannut pottiin.

`double getAmountRiseable()`
Palautetaan pelaajan enimmäiskorotus.

`double getAmountToCall()`
Palautetaan pelimerkkimäärä, joka pelaajan on maksettava pysyäkseen mukana.

`double getBankRoll()`
Palautetaan pelaajan pelimerkkimäärä.

`double getBankRollAtRisk()`
Palautetaan pelimerkkimäärä, joka on pelaajan oma pelimerkkimäärä, jos toisella jaossa mukana olevalla pelaajalla on enemmän merkkejä tai muista mukana olevien pelaajien pelimerkkimääristä suurin, jos pelaajalla on enemmän merkkejä kuin muilla mukana oli-

joilla.

`double getBankRollAtStartOfHand()`
Palautetaan pelimerkkimäärä, joka pelaajalla oli jaon alussa ennen sokko -tai aloituspanoksien maksamista.

`double getBankRollInSmallBets()`
Palautetaan pelaajan pelimerkkimäärä isoina sokkopanoksina.

`GameInfo getGameInfo()`
Palautetaan GameInfo-olio, joka määrittelee jaon tilatiedon pelaajan viimeisen toiminnon aikana.

`int getLastAction()`
Palautetaan kokonaisluku, joka vastaa pelajan viimeistä toimintoa. Palautettava on Holdem-luokassa määritelty vakio FOLD, CALL tai RAISE.

`java.lang.String getName()`
Palautetaan pelaajan nimen merkkijonoesitys.

`double getNetGain()`
Palautetaan pelimerkkimäärä, jonka pelaaja on voittanut tai hävinnyt jaon alettua.

`double getRaiseAmount(double amountToRaise)`
Palautetaan pelaajan korotuksen enimmäismäärä. Parametrina saadaan pelaajan pyytämä korotus. Palautettava määrä ei voi olla suurempi kuin pelaajan käytettävissä oleva pelimerkkimäärä.

`Hand getRevealedHand()`
Palautetaan pelaajan näyttämistä korteista muodostettu Hand-olio.

`int getSeat()`
Palautetaan pelaajan paikka.

`boolean hasActedThisRound()`
Palautetaan tosi, jos pelaaja on ollut jo vuorossa tällä kierroksella. Muuten palautetaan epätosi.

`boolean hasEnoughToRaise()`
Palautetaan tosi, jos pelaajalla on tarpeeksi pelimerkkejä korottaakseen. Muuten palautetaan epätosi.

`boolean inGame()`
Palautetaan tosi, jos pelaajalle on jaettu kortteja tässä jaossa. Muuten palautetaan epätosi.

`boolean isActive()`
Palautetaan tosi, jos pelaaja on vielä on vielä mukana kädessä. Muuten palautetaan epätosi.

`boolean isAllIn()`
Palautetaan tosi, jos pelaaja on ”all-in“. Muuten palautetaan epätosi.

`boolean isButton()`
Palautetaan tosi, jos pelaaja on jakajan paikalla. Muuten palautetaan epätosi.

`boolean isCommitted()`
Palautetaan tosi, jos pelaaja on sijoittanut rahaa tällä kierroksella. Muuten palautetaan epätosi.

`boolean isFolded()`
Palautetaan tosi, jos pelaaja on luopunut korteistaan. Muuten palautetaan epätosi.

`boolean isSittingOut()`
Palautetaan tosi, jos pelaaja on jättänyt jaon väliin. Muuten palautetaan epätosi.

`java.lang.String toString()`
Palautetaan pelaajan tilatiedon merkkijonoesitys.

ActionTypes

Kuvaus:
Luokassa ActionTypes on määritelty tunnukset Texas Hold ’em -pelipöydän eri toiminnoille.

Muuttujat:
`static final int ALLIN_PASS`
Pelaaja on pannut peliin kaikki pelimerkkinsä ja siten jättää väliin vuoron.

`static final int BET`
Panostaminen

`static final int BIG_BLIND`
Ison sokkopanoksen maksaminen

`static final int CALL`
Maksaminen

`static final int CHECK`

Passaaminen

`static final int FOLD`
Korteista luopuminen

`static final int INVALID`
Epäkelpo(sic) toiminto

`static final int MUCK`
Käden luovuttaminen näyttämättä kortteja

`static final int POST_ANTE`
Aloituspanoksen maksaminen

`static final int POST_BLIND`
Ison sokkopanoksen maksaminen peliin liityttäessä

`static final int POST_DEAD_BLIND`
Pelaajan jätettyä jakoja väliin kesken pelin häneltä ohi menneiden sokkopanosvuorojen maksaminen `static final int RAISE`
Korottaminen

`static final int SIT_OUT`
Jaon jättäminen väliin

`static final int SMALL_BLIND`
Pienen sokkopanoksen maksaminen

`static final int ENTER_TABLE`
Pelipöytään liittyminen

`static final int LEAVE_TABLE`
Pelipöydästä poistuminen

`static final int SIT_DOWN`
Pelipöytään istuminen

`static final int SIT_UP`
Pelipöydästä nouseminen

Action

Kuvaus:
Luokka Action määrittelee pelaajan pokeripelissä tekemät toiminnot. Toimintotyypit per-

itään ActionTypes-luokalta.

Muuttujat:

`private int type`
Action-olion tyyppi, joka on jokin edellä ActionTypes-luokassa määritellyistä

`private double toCall`
Maksamisen hinta

`private double amount`
Korotuksen määrä

Metodit:
`Action(int type, double toCall, double amount)`
Luodaan Action-olio antamalla parametreina toiminnon tyyppi, joka on yksi luokan muuttujista, maksamiseen vaadittava pelimerkkimäärä sekä maksun lisäksi sijoitettava pelimerkkimäärä.

`static Action allInPassAction()`
Palautetaan Action-olio, jonka tyyppi on ALLIN_PASS.

`static Action betAction(double amountToRaise)`
Palautetaan Action-olio, jonka tyyppi on BET. Panostuksen koko määräytyy parametrina saadun pelimerkkimäärän perusteella.

`static Action betAction(GameInfo gi)`
Palautetaan Action-olio, jonka tyyppi on BET. Panostuksen koko määräytyy parametrina saadun GameInfo-olion perusteella.

`static Action bigBlindAction(double toPost)`
Palautetaan Action-olio, jonka tyyppi on BIG_BLIND. Parametrina saadaan ison sokkopanoksen koko.

`static Action callAction(double toCall)`
Palautetaan Action-olio, jonka tyyppi on CALL. Parametrina saadaan maksamiseen vaadittava pelimerkkimäärä.

`static Action callAction(GameInfo gi)`
Palautetaan Action-olio, jonka tyyppi on CALL. Maksamisen hinta määräytyy parametrina saadun GameInfo-olion perusteella.

`static Action checkAction()`
Palautetaan Action-olio, jonka tyyppi on CHECK.

`static Action checkOrFoldAction(double toCall)`
Palautetaan Action-olio, jonka tyyppi on on CHECK, jos passaaminen on mahdollista, ts. parametrina saatu toCall = 0. Muuten palautettavan Action-olion tyyppi on FOLD.

`static Action checkOrFoldAction(GameInfo gi)`
Palautetaan Action-olio, jonka tyyppi on CHECK, jos passaaminen on mahdollista, ts. parametrina saadun GameInfo-olion perusteella saatu maksaminen hinta = 0. Muuten palautettavan Action-olion tyyppi on FOLD.

`Action equivalent(Action a)`
Palautetaan tosi, jos parametrina saatu Action-olion muuttujat type, toCall ja amount ovat kukin yhtä suuria kutsuvan Actionin muuttujien kanssa.

`static Action foldAction(double toCall)`
Palautetaan Action-olio, jonka tyyppi on FOLD. Parametrina saadaan maksamiseen vaadittava pelimerkkimäärä.

`static Action foldAction(GameInfo gi)`
Palautetaan Action-olio, jonka tyyppi on FOLD. Maksamisen hinta määräytyy parametrina saadun GameInfo-olion perusteella.

`static String formatCash(double value)`
Metodi toteutetaan tynkänä yhteensopivuuden vuoksi. Metodin kutsuminen aiheuttaa ajonaikaisen poikkeuksen, Unsupported Operation Exceptionin.

`static String formatCashFulol(double value)`
Metodi toteutetaan tynkänä yhteensopivuuden vuoksi. Metodin kutsuminen aiheuttaa ajonaikaisen poikkeuksen, Unsupported Operation Exceptionin.

`static Action getAction(int a, double toCall, double amount)`
Palautetaan saatujen parametrien perusteella luotava Action-olio. Parametri a määrää, onko palautettavan Action-olion tyyppi FOLD, CALL vai RAISE, toCall on maksamiseen vaadittava pelimerkkimäärä ja amount on korotettava pelimerkkimäärä (tai 0).

`int getActionIndex()`
Palautetaan kokonaisluku kutsuvan Action-olion tyypin perusteella. Jos tyyppi on FOLD, CALL tai RAISE, palautetaan vastaava HOLDEM-luokan määrittelemä vakio. Jos tyyppi ei vastaa tavallista vapaaehtoista toimintoa, palautetaan -1.

`double getAmount()`
Palautetaan amount-muuttujan arvo.

`double getToCall()`
Palautetaan toCall-muuttujan arvo.

`int getType()`
Palautetaan type-muuttujan arvo.

`boolean isAllInPass()`
Palautetaan tosi, jos kutsujan tyyppi on ALLIN_PASS. Muuten palautetaan epätosi.

`boolean isAnte()`
Palautetaan tosi, jos kutsujan tyyppi on POST_ANTE. Muuten palautetaan epätosi.

`boolean isBet()`
Palautetaan tosi, jos kutsujan tyyppi on on BET. Muuten palautetaan epätosi.

`boolean isBetOrRaise()`
Palautetaan tosi, jos kutsujan tyyppi on BET tai RAISE. Muuten palautetaan epätosi.

`boolean isBigBlind()`
Palautetaan tosi, jos kutsujan tyyppi on BIG_BLIND. Muuten palautetaan epätosi.

`boolean isBlind()`
Palautetaan tosi, jos kutsujan tyyppi on BIG_BLIND tai SMALL_BLIND. Muuten palautetaan epätosi.

`boolean isCall()`
Palautetaan tosi, jos kutsujan tyyppi on CALL. Muuten palautetaan epätosi.

`boolean isCheck()`
Palautetaan tosi, jos kutsujan tyyppi on CHECK. Muuten palautetaan epätosi.

`boolean isCheckOrCall()`
Palautetaan tosi, jos kutsujan tyyppi on on CHECK tai CALL. Muuten palautetaan epätosi.

`boolean isFold()`
Palautetaan tosi, jos kutsujan tyyppi on FOLD. Muuten palautetaan epätosi.

`boolean isFoldOrMuck()`
Palautetaan tosi, jos kutsujan tyyppi on FOLD tai MUCK. Muuten palautetaan epätosi.

`boolean isMuck()`
Palautetaan tosi, jos kutsujan tyyppi on MUCK. Muuten palautetaan epätosi.

`boolean isPost()`
Palautetaan tosi, jos kutsujan tyyppi on POST_ANTE, POST_BLIND tai POST_DEAD_BLIND. Muuten palautetaan epätosi.

`boolean isPostDeadBlind()`
Palautetaan tosi, jos kutsujan tyyppi on POST_DEAD_BLIND. Muuten palautetaan epätosi.

`boolean isRaise()`
Palautetaan tosi, jos kutsujan tyyppi on RAISE. Muuten palautetaan epätosi.

`boolean isSitOut()`
Palautetaan tosi, jos kutsujan tyyppi on SIT_OUT. Muuten palautetaan epätosi.

`boolean isSmallBlind()`
Palautetaan tosi, jos kutsujan tyyppi on SMALL_BLIND. Muuten palautetaan epätosi.

`boolean isVoluntary()`
Palautetaan tosi, jos kutsujan tyyppi on FOLD, CALL tai RAISE. Muuten palautetaan epätosi.

`static Action muckAction()`
Palautetaan Action-olio, jonka tyyppi on MUCK.

`static Action postAnte(double toPost()`
Palautetaan Action-olio, jonka tyyppi on POST_ANTE. Aloituspanoksen koko saadaan parametrina.

`static Action postBlindAction(double toPost)`
Palautetaan Action-olio, jonka tyyppi on POST_BLIND. Sokkopanoksen koko saadaan parametrina.

`static Action postDeadBlindAction(double toPost)`
Palautetaan Action-olio, jonka tyyppi on POST_DEAD_BLIND. Sokkopanoksen koko saadaan parametrina.

`static Action raiseAction(double toCall, double amountToRaise)`
Palautetaan Action-olio, jonka tyyppi on RAISE. Maksamiseen vaadittava pelimerkkimäärä ja korotuksen koko saadaan parametreina.

`static Action raiseAction(GameInfo gi)`
Palautetaan Action-olio, jonka tyyppi on RAISE. Maksamiseen vaadittava pelimerkkimäärä ja korotuksen koko määräytyvät parametrina saadun GameInfo-olion perusteella.

`static Action raiseAction(GameInfo gi, double amountToRaise)`
Palautetaan Action-olio, jonka tyyppi on RAISE. Maksamiseen vaadittava pelimerkkimäärä määräytyy parametrina saadun GameInfo-olion ja korotuksen koko amountToRaise-parametrin perusteella.

`static Action sitout()`
Palautetaan Action-olio, jonka tyyppi on SIT_OUT.

`static Action smallBlindAction(double toPost)`
Palautetaan Action-olio, jonka tyyppi on SMALL_BLIND. Pienen sokkopanoksen koko saadaan parametrina.

`String toString()`
Palautetaan Action-olion merkkijonoesitys. Metodi kuormittaa class.java.lang.Object-luokalta perityn metodin.

`String toString2()`
Palautetaan Action-olion merkkijonoesitys.

Holdem

Kuvaus:
Luokka Holdemissa on määritelty Texas Hold'emissa käytettäviä vakioita.

Muuttujat:
`static int BET`
Vakioarvo panostamiselle

`static int CALL`
Vakioarvo maksamiselle

`static int CHECK`
Vakioarvo passaamiselle

`static int FLOP`
Vakioarvo toiselle panostuskierrokselle

`static int MAX_PLAYERS`
Vakioarvo pelaajien enimmäismäärälle

`static int MAX_RAISES`
Vakioarvo panostusikkunoiden enimmäismäärälle rajoitetussa Texas hold'em -pelissä

`static int PREFLOP`
Vakioarvo ensimmäiselle panostuskierrokselle

`static int RAISE`
Vakioarvo korottamiselle

`static int RIVER`
Vakioarvo neljännelle panostuskierrokselle

`static int SHOWDOWN`
Vakioarvo käden paljastamiselle

`static int TURN`
Vakioarvo kolmannelle panostuskierrokselle

7 Tietokannat

Kunkin pelimoduulin tarvitsemat taulut tehdään omaan skeemaansa. Käyttäjät lisätään rekisteröinnin yhteydessä palvelimen käyttäjätietokantaan ja ensimmäistä kertaa peliin liittymisen yhteydessä pelimoduulin käyttäjätauluun.

SQL-lauseet

Palvelimen käyttäjätietokanta

Palvelimen käyttäjätietokantaan eli automaattisesti luotavaan public-skeemaan liittyvät lauseet.

Taulun luominen:

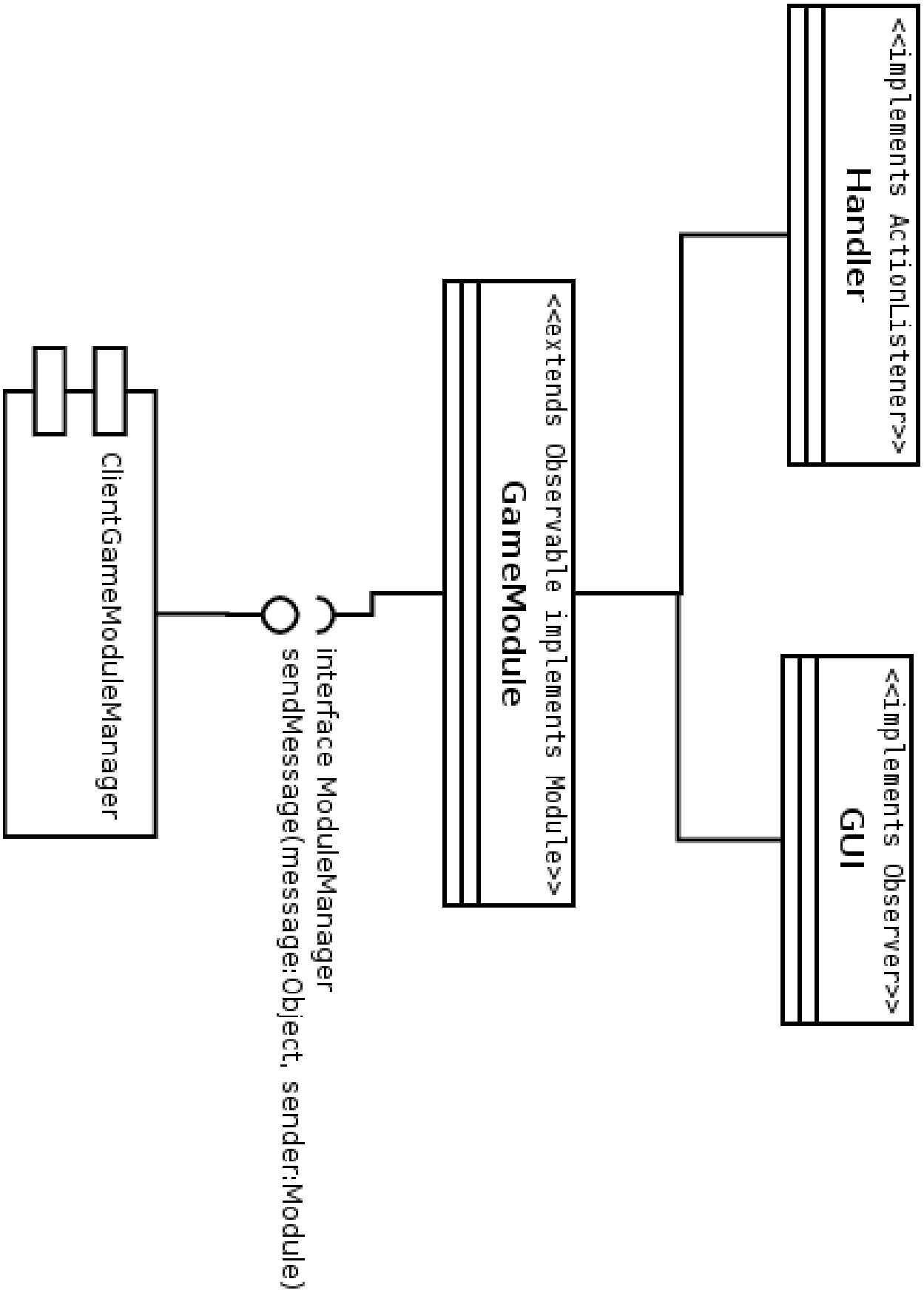
```
CREATE TABLE public.user (
username varchar(12) PRIMARY KEY,
password char(20) NOT NULL,
superuser boolean
);
```

Käyttäjän lisääminen (ensin tarkistettava, onko käyttäjä jo tietokannassa, kts. alempana):

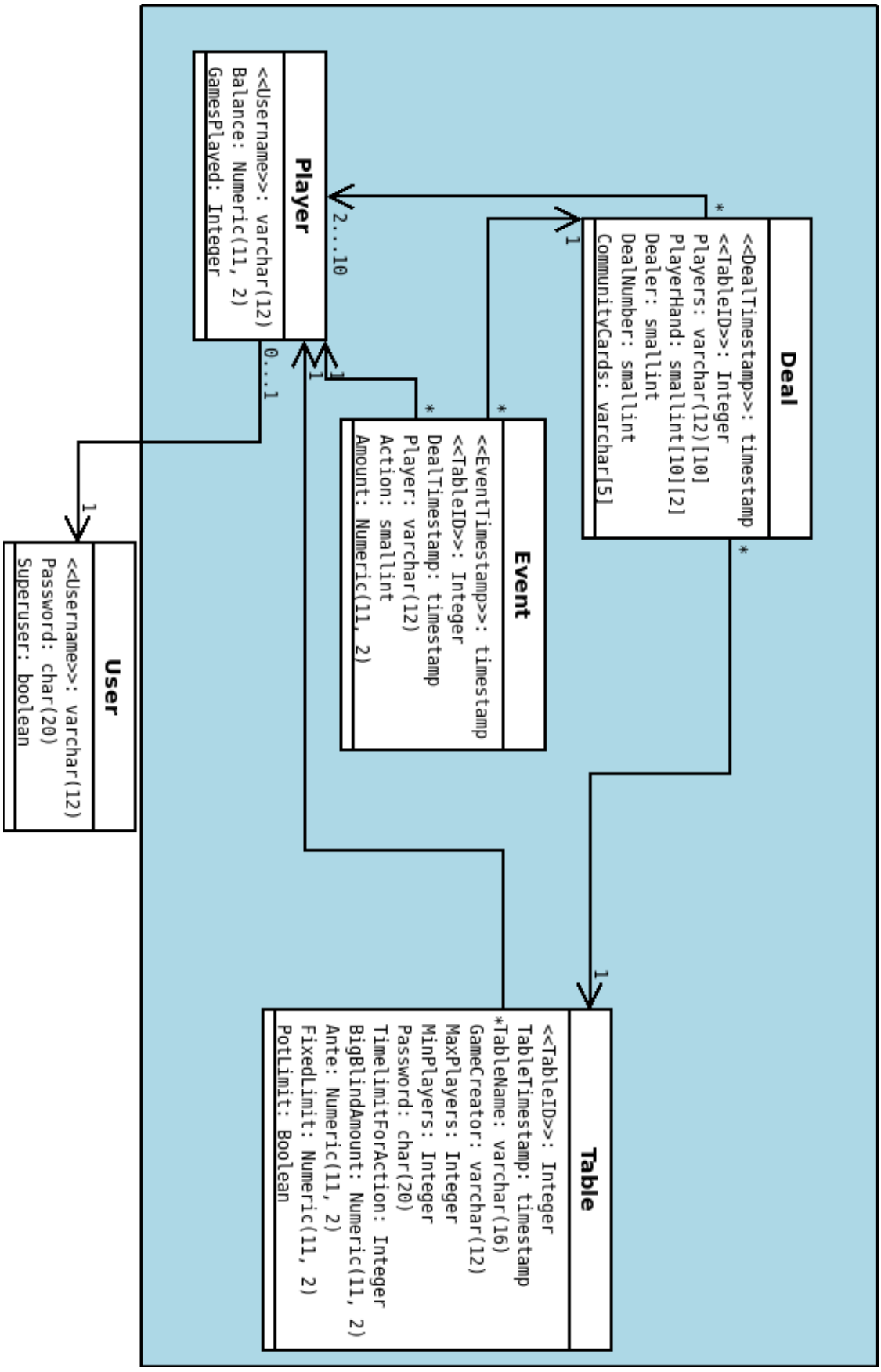
```
INSERT INTO public.user VALUES ('Pokeriv3', 'salasana', false);
```

Käyttäjän tunnistaminen käyttäjätietokannasta (annetut parametrit käyttäjätunnus "Pokeriv3"ja salasana "salasana"; jos kysely palauttaa luvun yksi, käyttäjä on tietokannassa):

```
SELECT count(*)
FROM public.user
WHERE username='Pokeriv3' AND password='salasana';
```



Kuva 15: Texas Hold'em pelimoduulin asiakasliittymä



Kuva 16: Relaatiokaavio palvelimen käyttäjätietokannasta ja TexasHoldem -pelimoduulin tietokannasta

TexasHoldem-pelimoduulin skeema

Muiden pelimoduulien skeemat voi piilottaa istunnon aikana asettamalla käyttäjälle roolin, jolloin hänellä on vain rooliin kuuluvat oikeudet.

TexasHoldem-skeemalle luodaan rooli komennolla:

```
CREATE ROLE texasholdem_role LOGIN;
```

Tarvittavien oikeuksien antaminen roolille:

```
GRANT USAGE ON SCHEMA texasholdem TO texasholdem_role;
GRANT ALL ON public.user, texasholdem.table, texasholdem.deal,
texasholdem.event, texasholdem.player TO texasholdem_role;
```

Oikeuksien poistaminen roolilta:

```
REVOKE USAGE ON SCHEMA texasholdem TO texasholdem_role;
REVOKE ALL ON public.user, texasholdem.table, texasholdem.deal,
texasholdem.event, texasholdem.player FROM texasholdem_role;
```

Roolin tuhoaminen:

```
DROP ROLE texasholdem_role;
```

Roolin jäsenyyden antaminen käyttäjälle:

```
GRANT texasholdem_role TO tkt_pok3;
```

Roolin asettaminen käyttäjälle niin, että käyttäjä pääsee istunnon aikana käsiksi vain roolin oikeuksien sallimiin tauluihin (palvelimen pitäisi tehdä tämä pelimoduulin tehdessä tietokantaoperaatioita):

```
SET ROLE texasholdem_role;
```

Käyttäjä saadaan takaisin alkuperäiselle oikeustasolle komennolla:

```
RESET ROLE;
```

Skeeman luonti:

```
CREATE SCHEMA texasholdem;
```

Taulujen luomiset:

```
CREATE TABLE texasholdem.player (
username varchar(12) PRIMARY KEY REFERENCES public.user (username),
balance numeric(11,2),
gamesplayed integer
);

CREATE TABLE texasholdem.table (
tableid integer PRIMARY KEY,
tabletimestamp timestamp,
tablename varchar(16),
gamecreator varchar(12) REFERENCES texasholdem.player (username),
maxplayers integer,
minplayers integer,
password char(20),
timelimitforaction integer,
bigblindamount numeric(11,2),
ante numeric(11,2),
fixedlimit numeric(11,2),
potlimit boolean
);
```

Funktio taulukossa olevien pelaajien käsittelyä varten (tarkistaa, ovatko taulukossa olevat pelaajat olemassa Player-pöydässä):

```
CREATE FUNCTION test_players (varchar(12)[]) RETURNS bool AS'
select case when count(1) = array_upper($1,1) then true
else false
end from texasholdem.player where username=any($1);
' LANGUAGE SQL;

CREATE TABLE texasholdem.deal (
dealtimestamp timestamp,
tableid integer REFERENCES texasholdem.table (tableid),
players varchar(20)[10] CHECK (test_players(players)),
playerhand smallint[10][2],
communitycards smallint[5],
dealer smallint,
```

```
dealnumber smallint,
PRIMARY KEY(dealtimestamp, tableid)
);

CREATE TABLE texasholdem.event (
eventtimestamp timestamp,
dealtimestamp timestamp,
tableid integer,
player varchar(12) REFERENCES texasholdem.player (username),
action smallint,
amount numeric(11,2),
PRIMARY KEY(eventtimestamp, tableid),
FOREIGN KEY (dealtimestamp, tableid) REFERENCES texasholdem.deal
(dealtimestamp, tableid)
);
```

Pelaajan lisääminen TexasHoldemiin:

```
INSERT INTO texasholdem.player VALUES ('Pokeriv3', 0, 0);
```

Käyttäjän tunnistaminen TexasHoldem-skeemasta (annettu parametri käyttäjätunnus "Pokeriv3"; jos kysely palauttaa luvun yksi, käyttäjä on skeemassa):

```
SELECT count(*)
FROM texasholdem.user
WHERE username='Pokeriv3';
```

Pöydän lisääminen (pöydän aikaleima luodaan samalla kuin itse pöytä):

```
INSERT INTO texasholdem.table VALUES (13, CURRENT_TIMESTAMP,
'Kairo', 'Pokeriv3', 10, 2, 'salasana', 90, 5000, 1000, 100000,
true);
```

Jaon lisääminen (pelimoduuli vastaa tietokantaan tallennettavan tiedon sisällöstä):

```
INSERT INTO texasholdem.deal (dealtimestamp, tableid, players,
playerhand, communitycards, dealer, dealnumnber)
VALUES ('2008-03-13 06:46:11.887628', 13,
ARRAY['Pokeriv3', 'Richelieu', 'Leonardo'],
ARRAY[[2,3], [10,40], [32,12]], ARRAY[4,17,51,20], 2, 2);
```

Tapahtuman lisääminen:

```
INSERT INTO texasholdem.event
(eventtimestamp, tableid, dealtimestamp, player, action, amount)
VALUES ('2008-03-13 06:47:13.839455', 13,
'2008-03-13 06:46:11.887628', 'Pokeriv3', 2, 1000.00);
```

Tallennettua pelilokia syötettäessä tietokantaan käytetään ylläolevia lauseita kunkin jaon ja sen tapahtumien tallentamiseen. Lisäksi peliin osallistuneiden pelaajien balance on päivitettävä (korvaa parametrit Pokeriv3 kyseisen pelaajan tunnuksella ja 1000 voitettujen pelimerkkien määrällä):

```
UPDATE texasholdem.player SET balance = ((SELECT balance
FROM texasholdem.player
WHERE username = 'Pokeriv3') + 1000)
WHERE username = 'Pokeriv3';
```

Pelipöydän hakeminen TexasHoldem-skeemasta:

```
SELECT tablename,
CASE WHEN password IS NOT NULL THEN 'true'
ELSE 'false'
END AS ispasswordprotected,
(maxplayers - array_upper(players, 1)) AS seats_left,
timelimitforaction, bigblindamount, ante, fixedlimit, potlimit
FROM texasholdem.table, texasholdem.deal
WHERE texasholdem.table.tableid=texasholdem.deal.tableid;
```

Esimerkki pelipöydän fixedlimitin päivittämisestä:

```
UPDATE texasholdem.table SET fixedlimit = 1000000
WHERE tableid=13;
```

Jako-otteen kysyminen tietokannasta (tiedettävä pöydän id ja jaon aika):

```
SELECT DISTINCT tablename, dealnumber, array_upper(players, 1),
dealer, bigblindamount, potlimit, fixedlimit, ante, players,
playerhand, communitycards
FROM texasholdem.deal, texasholdem.table
WHERE texasholdem.table.tableid=13
AND texasholdem.deal.tableid=13
AND texasholdem.deal.dealtimestamp='2008-03-13 06:46:11.887628';
```

Jaon pelaajamäärän kysyminen:

```
SELECT array_upper(players, 1)
FROM texasholdem.deal
WHERE tableid=13 AND dealtimestamp='2008-03-13 06:46:11.887628';
```

Jaon korttien kysyminen:

```
SELECT playerhand, communitycards
FROM texasholdem.deal
WHERE tableid=13 AND dealtimestamp='2008-03-13 06:46:11.887628';
```

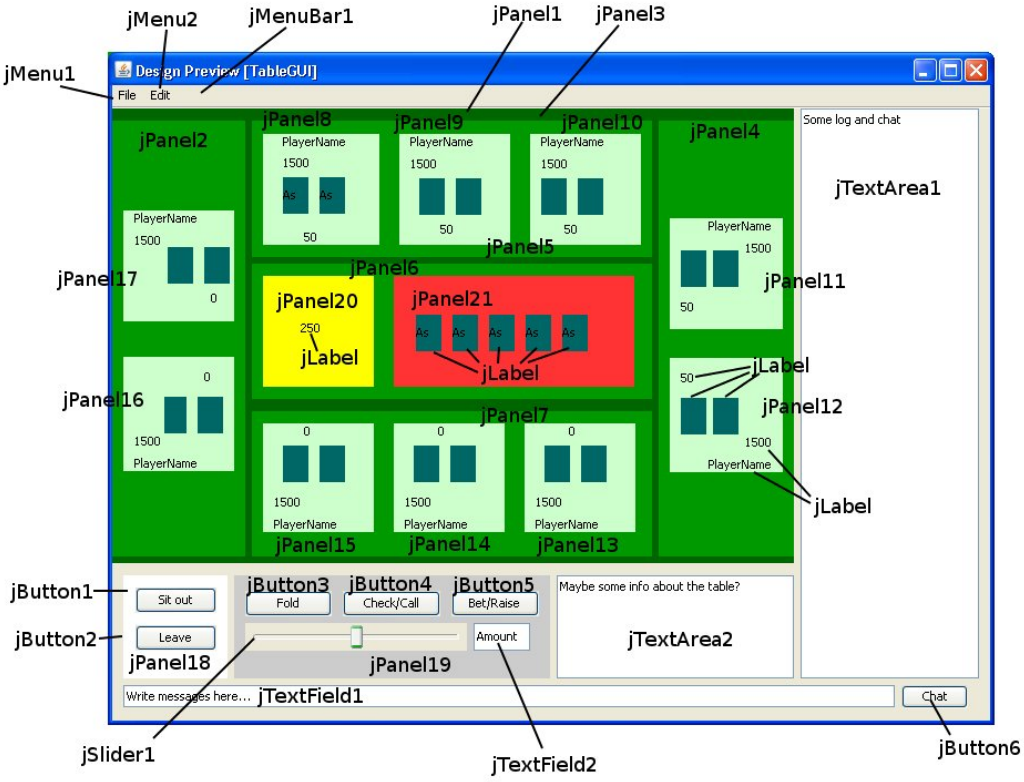
Voittotietojen tulostus (vain TexasHoldem-tietokannasta) (KT17):

```
SELECT public.user.username, gamesplayed, balance
FROM public.user, texasholdem.player
WHERE public.user.username=texasholdem.player.username;
```

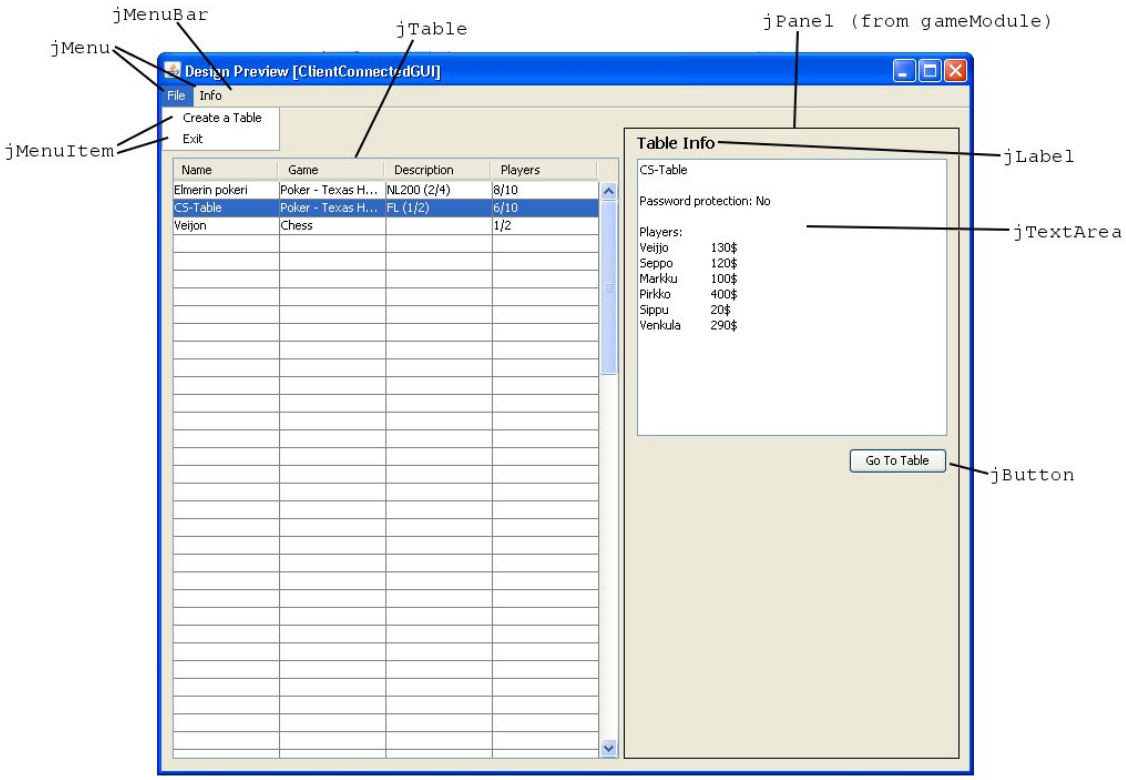
Voittojen määrä jaoista, joihin pelaaja on osallistunut (korvaa muuttuja VOITTO voitto-actionin numerolla ja Pokeriv3 pelaajan tunnuksella) (KT11):

```
SELECT wins/alldeals
FROM (SELECT count(deal.dealtimestamp) AS wins
FROM texasholdem.deal, texasholdem.event

WHERE action=VOITTO AND 'Pokeriv3' = ANY (deal.players)) winstable,
(SELECT count(deal.dealtimestamp) AS alldeals
FROM texasholdem.deal
WHERE 'Pokeriv3' = ANY (deal.players)) alldealstable;
```



Kuva 17: Texas Hold’em pöytänäkymä



Kuva 18: Aulanäkymä