SOFTWARE DISTRIBUÏT

Pràctica 1

Adrián Pardos López

Carles Planella Gómez

ÍNDEX

Introducció

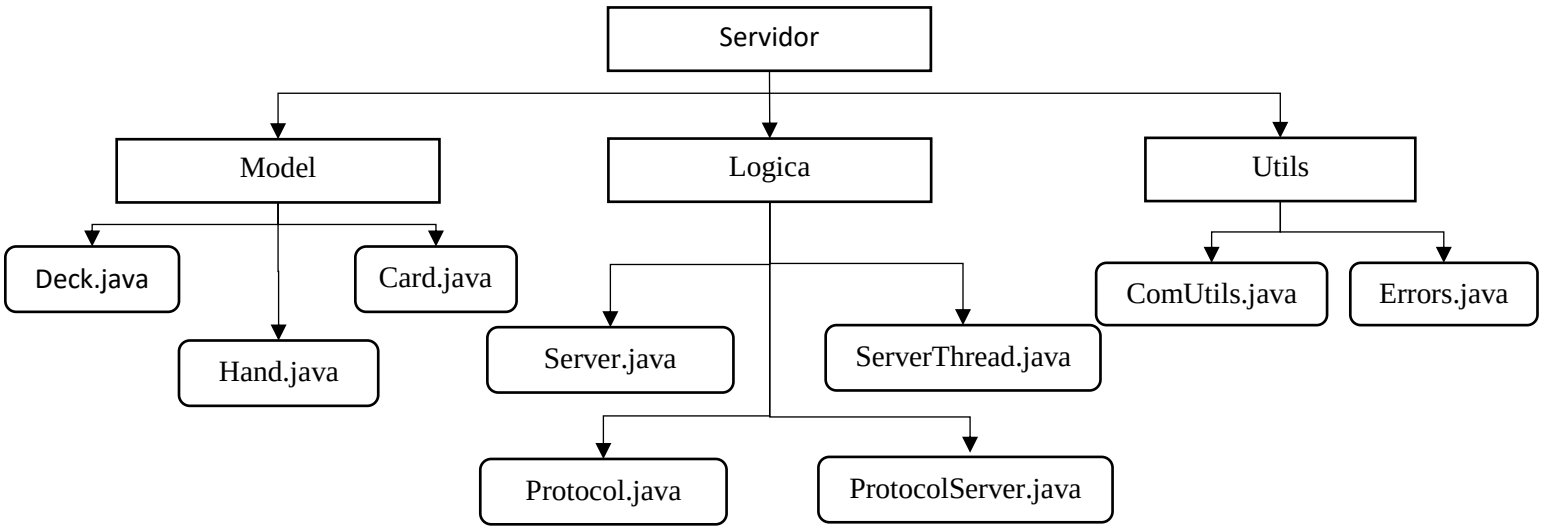
Aquesta pràctica tracta de crear el joc de pòquer estàndard usant sockets en el llenguatge Java. Per això hem implementat un model amb 2 projectes ben diferenciats. Un client i un servidor. Tots dos son capaços de comunicar-se entre si, mitjançant una adreça i un port. Tot client haurà de connectar-se a en aquesta adreça i en aquest port per establir connexió i així poder jugar.

Tota aquesta connexió ha de ser tractada en un protocol, tant en el projecte *Servidor* com en el *Client*. A més a més, com a requisits del projecte tenim que el Servidor ha de tenir una intel·ligència artificial, i el Client ha de ser tant manual, on un usuari podrà interactuar i jugar, com amb una intel·ligència.

Servidor

Introducció al servidor

Distribució del projecte:



Com podem veure hem distribuït les classes en diferents packages. El primer Package tenim les cartes, la baralla i la mà, classes necessàries per un joc de cartes. Al Package Logica tenim tota la lògica del joc i de la connexió. Per últim, a Utils tenim diferents funcions per poder comunicar-nos amb el client o enviar errors

Package Model:

Recordem que estem utilitzant programació orientada a objectes. En aquest Package tenim les classes necessàries per tal de poder realitzar una partida de pòquer, o que podríem necessitar per a qualsevol joc de cartes.

- Deck.java: Aquesta classe ens permet tenir la informació de la baralla. Es crea un nou *Deck* cada cop que es comença una nova partida. Conté també els mètodes necessaris per tal de gestionar la baralla.-

- Hand.java: Aquesta classe conté la informació de la mà, tant del servidor com la del client. Conté els mètodes necessaris per tal de gestionar la mà. A més a més, conté un mètode que ens permet saber quin és el valor d'una mà en un moment concret, el qual ens ajuda, al final d'una partida, a veure quina mà pot ser la millor.
- Card.java: Conté la informació necessària d'un objecte de tipus carta, com per exemple el seu rank o suit. També tenim els mètodes necessàries per tal de gestionar aquest objecte tipus carta, com obtenir el rank d'una carta.

  **Per més informació sobre variables o mètodes, consultar el javadoc.*

Package Logica:

Aquest Package és molt important, ja que conté tota la lògica del servidor, així com la gestió de la connexió i la gestió de diferents fils, que utilitzem per tal de poder tenir més d'un client connectat a l'hora al nostre servidor.

- Server.java: Main del projecte. Tenim definit el port pel qual els clients es connectaran així com la creació del socket i la creació de diferents threads. El que fem per poder gestionar els threads i les connexions dels clients, és tenir un bucle infinit on creem un socket i fem un accept(), que el que fa és esperar la connexió d'un nou client. A continuació es crea un thread, i fem un start(), per tal de cridad el mètode run() de la classe *ServerThread.java* i poder començar una nova partida, amb un thread diferent.

- ServerThread.java: Conté la informació necessària per tal d'iniciar la connexió amb un client a traves d'un thread. Aquesta classe inicia el protocol i la partida. També es crea un fitxer .log per tal de guardar la partida, que es guarda a la carpeta del projecte per defecte.

- Protocol.java: En aquesta classe tenim tota la lògica del servidor. Primer de tot es configura la partida assignant fitxes al client i el servidor, i assignant una valor com aposta mínima. Veurem amb més detall aquesta classe més endavant.

- ProtocolServer.java: És una classe que ajuda al Protocol principal a realitzar les seves funcions. Tenim diferents mètodes per gestionar diferents moments de la partida, com podria ser enviar un STKS, fer un showdown o que el server generi una opció aleatòria (IA bàsica). També la veurem amb més detall a continuació.

**Per més informació sobre variables o mètodes, consultar el javadoc.*

Package Utils

Aquest Package s'encarrega de controlar l'aplicació.

- ComUtils.java: Classe proporcionada a l'enunciat. S'han creat nous mètodes per tal de fer més senzilla la comunicació amb el client.

- Errors.java: Classe que s'encarrega d'enviar errors, actualment con´te tres errors principals, DataError, CommandError i NotChips.

**Per més informació sobre variables o mètodes, consultar el javadoc.*

Modificacions al ComUtils.java

Per tal de fer més senzilla la utilització del buffer per passar informació als diferents clients que tenim connectats, s'han afegit alguns mètodes nous.

- write_SP(): Aquest mètode s'encarrega d'enviar 1 char (que serà 1 byte), el qual conté un espai.

- string_to_buffer(): Permet escriure un String d'una mida X qualsevol al buffer.

- int_to_buffer(): Permet escriure un int (4 bytes) al buffer.

- write_buffer(): Permet enviar un array de bytes.

Aquest mètodes ens han facilitat molt la programació i gestió de les dades a comunicar.

Lògica del protocol (Protocol.java i ProtocolServer.java)

Per tal de realitzar la lògica del servidor, teníem uns exemples i unes instruccions, les quals s'han seguit per tal de realitzar i gestionar de la manera indicada.

Com hem dit a la introducció del servidor, el primer que fem es assignar l'aposta inicial, les fitxes al client i al servidor, escriure la informació al log i cridar a un mètode *partida()* que inicia realment el protocol.

El primer es enviar un ANTE al client indicant quina és l'aposta mínima per tal d'entrar a la partida i la un STKS, que conté les monedes actuals del client i del servidor.

El client haurà de dir, amb un ANOK, que accepta les condicions de la partida, i per tant podrem començar. Cal dir que es fa un control de la comanda enviada pel client, el que fem és posar en majúscules la comanda que ha introduït per evitar errors.

Quan el client ens confirma que accepta, es comprova si es té, tant el client com el servidor, la quantitat de fitxes per tal de poder iniciar la partida. Aquesta quantitat bé definida per l'aposta inicial que s'ha assignat abans de començar la partida. En el cas de que no es pugui començar la partida per qüestions de fitxes, s'enviarà un sendChipError.

<u>Cas ANOK:</u>

Si es pot començar la partida, es genera un nou Deck, es fan les primeres gestions de diners (restar l'aposta inicial de les monedes del client i del servidor) i s'escriu la informació al log. També assignem 5 cartes a la mà del client i del servidor, diem qui és el dealer. Al client li enviem el dealer i la seva mà. En el cas de que el dealer sigui 1, significa que és el server qui ha d'iniciar la partida, per tant escollirem una opció aleatòria cridant a un mètode que tenim a *ProtocolServer.java*, que el que fa es generar una opció al atzar segons el moment en el que estiguem. En aquest cas, quan el client ens fa un ANOK i li toca fer una acció al server, podem fer o un *PASS*, o un *BET*. En cas de fer un

BET, cridem al mètode serverBet que també tenim a ProtocolServer.java, el qual fa una aposta aleatòria, de entre 0-100 fitxes.

Cas PASS:

En el cas de que el client ens faci un PASS, haurem de gestionar els diferents casos, segons el moment de la partida en el que estiguem.

Tenim dos casos, el primer cas controla si el dealer es el servidor i si ja hem fet el descartar cartes, si es compleixen aquestes dues condicions, significa que la partida acaba, i per tant hem d'enviar un SHOW i un STKS. L'altre cas, en el cas que el dealer sigui el client, li tocarà al servidor generar una resposta, la qual com opcions té fer també un *PASS* o fer un *BET.*

En el cas de realitzar un *PASS,* comprovarem si ja hem descartat. Si hem fet ja el *DRAW*, significa que s'ha de finalitzar la partida i hem d'enviar un *SHOW* i un *STKS.*

Cas CALL:

En el cas de que el client ens faci un *CALL*, haurem de gestionar les fitxes, la qual cosa podem fer gracies a una variable que es diu LastBet de ProtocolServer.java que controla quin ha sigut el valor de l'última aposta.

En el cas de que ja haguem eliminar cartes (DRAW), haurem de finalitzar la partida enviant abans un *SHOW* i un *STKS.*

Cas RISE:

Abans de entrar en la lògica, dir que per tal de llegir el valor de la pujada que ha fet al server, al mètode readbytes de la classe ComUtils, li hem de passar un "be", per tal de dir-li que estem utilitzant big endian.

En el cas del *RISE*, tenim dues condicions, si ja hem fet el *DRAW* o no. En el cas de que ja l'hem fet, haurem de generar una decisió. En el cas que no un altre. La diferencia entre les dues, és que en el cas de ja tenir fet el *DRAW*, la partida pot acabar i per tant li hauríem d'enviar al client un *SHOW* i un *STKS*. En el cas de ser un *FOLD* no s'envia el *SHOW*.

Les opcions que tenim per fer la decisió del servidor son *CALL o FOLD.*

Cas BET :

Aquí no hem de controlar a quin moment de la partida estem, ja que si el client ens fa un *BET_* significa que li hem de contesta, per tant hem de generar una opció aleatòriament. Entre aquestes opcions tenim *CALL,FOLD* i *RISE.*

Cas DRAW:

El primer que fem en entrar a aquest cas, és posar el valor del booleà que tenim a ProtocolServer.java que gestiona si ja hem fet el draw o no a true.
Llavors hem de llegir quin es el nombre de cartes que el client vol treure de la seva mà per tal de poder fer un for i per llegir cadascuna de les cartes i eliminar-les de la seva mà. Hem de controlar el cas de que sigui un 10, ja que en aquest cas haurem de llegir 3 bytes i no 2, com passa a les demes cartes que tenim. Comprovem si te aquesta carta i eliminem.

A continuació cridem a un mètode de *ProtocolServer* que fa que el server elimini aleatòriament un número de cartes de la seva mà.

En el cas de que el dealer sigui 1, és el servidor el que a part de enviar un *DRWS,* hagi d'enviar la seva decisió a jugar. Com a decisions, en aquest cas, tenim *PASS, BET_* o *FOLD.*

Cas FOLD:

En el cas de que el client faci un *FOLD,* es finalitza la partida amb el server com a guanyador.

Cas ERRO:
En el cas de rebre un error, l'haurem de tractar i escriure en el log.

Cas QUIT:

Si el client fa un *QUIT*, vol dir que no vol continuar la partida, i per tant hem de sortir, tancant el fitxer que estem utilitzant per escriure el log.

Funcions a destacar.

Per tal de poder fer la lògica s'han fet altres mètodes que ens permeten gestionar així la partida.
El mètode que cal recalcar es el WhoWins() de la classe ProtocolServer.java.
Aquest mètode permet decidir qui ha guanyat la partida, un cop finalitzada. Aquest mètode es crida al fer un *SHOW*, ja que és quan s'ha de tomar la decisió de qui és el guanyador de la partida.

EL que fa és cridar a un altre mètode que no tenim a la classe hand que dona un valor X a la mà, la qual ens permet saber qui ha sigut el guanyador. Per tal de fer més senzilla la comparació entre cartes i saber que és el que te aquella mà, s'ha ordenat la mà abans de tot.

Per tant, es crida aquest mètode per a les dues mans, servidor i client, i es fa una comparació del valor que retorna la funció, el que tingui el valor mes petit guanya. En el cas de tenir el mateix valor, s'ha fet una gestió de qui té la carta més alta, i així poder decidir qui guanya.
El valor de les decisions el podem veure en aqueta taula:

1	**Escala de color**
2	**Pòquer**
3	**Full**
4	**Color**
5	**Escala**
6	**Trio**
7	**Doble Parella**
8	**Parella**
9	**Carta més alta**

Persistència de dades i control de comandes

Per tal de millorar el protocol, hi ha persistència de dades. Consisteix que a partir del id, tenim una *Hash table* que ens permet veure si aquest client ja ha estat jugant, per tal de donar al client les monedes que tenia. Si es tanca el server la hash es perd (no hi ha persistència de dades a memòria), i per tant tots els clients també ja que no es guarden a memòria actualment.

Pel que fa el control de comandes, si un client introdueix una comanda incorrecte, s'envia un error i es tanca la connexió amb aquell client, per tant si vol tornar a jugar haurà de començar. Com que tenim persistència de dades, el client continuarà amb les mateixes monedes que tenia abans de cometre aquest error.

Per últim, per tal d'evitar més errors en les comandes, després de la sessió de test vam posar al protocol un delay, per evitar una saturació per part d'un client per exemple.
Aquest delay, en realitat es un sleep de 100 ms.

Client

Introducció al Client

El projecte Client es aquell que ens permet comunicar-nos amb el servidor i en el que l'usuari utilitzarà per jugar.
Aquest consta de les següents classes distribuïdes en un mateix Package:

- Client(main)
- ClientProtocol
- ClientIAProtocol
- ProtocolFunctions
- Player
- DataServer
- Errors
- ComUtils

Expliquem detalladament que fan aquests classes i perquè s'ha distribuït d'aquesta manera:

Classe Client

Aquesta classe conte el main del programa mitjançant el qual l; aplicació serà executada. Aquesta classe crea el socket principal de connexió mitjançant l'adreça IP servidora i el port que utilitzarà per a la comunicació. Aquest client a més s'ha implementat, tal que, alhora d'executar-lo per consola, puguem escollir el mode interactiu en que es vol que s'executi es a dir si en mode Manual o en mode Automàtic. Per això hem afegit com diu l'enunciat a la practica la incorporació d'un paràmetre opcional. En cas de no ser entrat aquest paràmetre, s'executarà en mode manual.

Un cop creat el socket, creem un objecte ClientProtocol o un objecte tipus ClientIAProtocol amb el socket de paràmetre, depenent el mode escollit.

Classe ClientProtocol

Aquesta classe es la que s'encarrega de distribuir les diferents opcions principals al usuari i també s'encarrega de rebre les opcions del servidor. Aquesta classe principalment consta de 4 mètodes importants, aquests són:

- ClientProtocol: És el constructor principal de la classe ClientProtocol en el que crea variïs objectes per anar gestionant la informació que va enviant i rebent.
- doProtocol: Aquest es un mètode que esta en bucle per mostrar la informació de la partida i per anar seleccionant alguna de les opcions disponibles.
- commandManager: Aquest és un mètode que s'encarrega de gestionar la comanda d'entrada escrita per l'usuari.
  Segons la comanda d'entrada, mitjançant un switch fem una cosa o un altre. Les opcions que tenim disponibles a fer per el servidor son les següents:
  - START: Envia un STRT amb el id al servidor. Es llegeix la resposta del servidor en aquest cas un ANTE y un STKS.

- ANTE_OK: Envia un ANOK al servidor, que confirma l'aposta mínima que ofereix el servidor. Després de haver enviat, rebem el DEAL i un HAND.
- QUIT: Amb aquesta instrucció el que fem es tancar connexió de socket i aturar l'execució del programa.
- BET: Aquesta comanda ens permet realitzar un BET d'una quantitat de diners al servidor. Després d'aquesta comanda esperem rebre la resposta del servidor, un CALL un RAISE o rarament un FOLD.
- PASS: Aquesta instrucció es per dir que no volem apostar per tant passem i esperem la resposta del servidor, que pot ser un BET, un PASS o rarament un FOLD. Depenent de qui sigui el dealer rebrem o no resposta del servidor.
- CALL: S'utilitza per igualar una aposta que ens hagi fet el servidor, per exemple si ell fa un BET 50 i nosaltres fem CALL, el que fem es igualar aquesta aposta i per tant apostem 50 monedes. Després d'aquesta instrucció, si hem descartat, possiblement rebem un SHOWDOWN i un STAKES.
- RAISE: Aquesta instrucció ens permet pujar una aposta que hàgim rebut. Després de realitzar la aposta rebrem una comanda per part del servidor per si vol o no igualar o pujar-la.
- FOLD: Serveix per plantar-nos en mig de una partida, i per tant automàticament perdem la partida. Després d'aquesta comanda es rep per part del servidor un STAKES.
- DRAW: Aquesta comanda ens permet descartar de 0 a 5 cartes de la ma que tinguem actualment. Per això deixem disponible al usuari el llistat de cartes per a que les seleccioni per a descartar-les.

Per a les instruccions anteriors, en cas de que vagi alguna cosa malament, enviem al servidor un error informant de l'error. Això ho fem mitjançant la classe Errors.

- serverResponseManagement: Aquest mètode gestiona les instruccions que venen per part del servidor. Per això el que es fa mitjançant la classe de ComUtils, llegim els primers 4 bytes per a saber la instrucció que rebem i que hem de gestionar. Les diferents instruccions que podem trobar per a gestionar son les següents:
  - ANTE: Amb aquesta instrucció rebem l'aposta inicial que hem de realitzar.
  - STKS: Rebem el estat actual de les monedes a la partida. En aquest cas, rebem les monedes corresponents al client i les monedes del servidor.
  - DEAL: Amb aquesta el que fem es llegir per dir-ho d'una manera es la prioritat en que es realitzen les apostes, es a dir si es 1, el servidor es el que tindrà prioritat, 0 tindrà el client.
  - HAND: Aquesta instrucció ens dona les cartes que inicialment tindrem a la partida.
  - BET_: Aquesta instrucció ens dona l'aposta que ha realitzat el servidor.
  - PASS: Aquesta instrucció ens indica que el servidor ha passat i depenent de qui sigui el dealer(prioritari) es farà una acció o una altre.

- CALL: Quan es rep aquesta instrucció vol dir que prèviament s'ha realitzat una aposta i per tant amb això el servidor el que ens diu es que iguala aquesta aposta.
- RISE: Quan es rep aquesta instrucció vol dir que prèviament s'ha realitzat una aposta i per tant amb això el servidor el que ens diu es que puja aquesta aposta amb una quantitat X de monedes.
- FOLD: Amb aquesta instrucció ens esta indicant el servidor que es planta i per tant perd la partida.
- DRWS: Aquesta instrucció retorna dues coses, primer retorna el mateix numero de cartes que el client ha descartat i seguidament d'haver enviat aquestes cartes, envia el numero de cartes que ha descartat el servidor.
- SHOW: Aquesta instrucció el que fa es mostrar la mà del Servidor un cop s'ha acabat la segona ronda. Llavors com s'ha dit anteriorment el servidor determina qui ha estat el guanyador. Depenent de això rebrem més o menys monedes de les que teníem inicialment mitjançant un missatge STKS.
- ERRO: Amb això rebem un error per part del servidor.

Un apunt, el client per tot error que enviï o rebi tenca la connexió amb el socket.

Classe ClientIAProtocol

Aquesta classe es idènticament igual a la classe ClientProtocol, excepte que en comptes de demanar dades de entrada a l'usuari, aquest es automàtic, per això s'ha creat una llista amb les opcions disponibles en cada moment. D'aquesta llista seleccionem una aleatòriament. Les quantitats d'apostes es determina mitjançant una formula molt senzilla, en el que manté una proporció segons la quantitat de diners disponible. La fórmula emprada per a qualsevol aposta és la següent:

$$Aposta \in \{min, max\} \mid \min = 1 \ , max = \frac{Balance}{3}$$

On Balance és la quantitat de diners del jugador.

El mètode de selecció de comandes es completament aleatori al igual que el numero de cartes que es descarten i les cartes que es descarten.

Classe ProtocolFunctions

Aquesta classe es la que s'encarrega de fer la feina més pesada per dir-ho d'una manera. Es la que realment s'encarrega de enviar al socket la informació i també de rebre la informació que hi ha disponible en el socket. Aquesta classe no conte un constructor, es una classe de suport que utilitza ClientProtocol i ClientIAProtocol. És una classe que únicament conté mètodes per l'enviament de comandes i per llegir comandes per part del servidor.

Classe Player

Aquesta classe, es la classe jugador i s'encarrega de emmagatzemar la informació de tot un jugador durant la partida. Els seus atributs més importants són:

- id, es el numero que representa a aquell jugador;
- balance, es el numero que te el jugador.
- hand, que conte un objecte de tipus ArrayList<Card>, bàsicament, conté les cartes del jugador.

També conte mètodes per obtenir aquests atributs o simplement per printar-los per pantalla.

Classe DataServer

És un objecte que conté la informació bàsica del servidor i també de la partida. Trobem per exemple atributs que afecten al funcionament de la partida com per exemple si s'ha descartat o no a la partida, qui és prioritari en aquella ronda, etc

Classe Errors

Aquesta classe es la que conté els diferents errors a enviar al servidor, com per exemple si hi ha hagut un problema de dades, en aquest cas el client envia DataError, etc Aquesta classe esta implementada per enviar errors tant per el Client com el Servidor.

Clase ComUtils

És la mateixa que implementa el servidor. Explicada en el servidor.

Organització de classes

En el client, les classes no han estat agrupades en paquets, degut a que l'enunciat vol fer servir una nomenclatura d'execució que amb Paquets de Java no podríem aconseguir. En cas de agrupar-ho en Paquets, ho dividiríem de la següent Manera.

Model:

- Player
- DataServer

Logica:

- Client(main)
- ClientProtocol
- ClientIAProtocol
- ProtocolFunctions

Utils:

- Errors
- ComUtils

Execució de Servidor

Per tal d'executar el servidor el que hem de fer es obrir l'entorn de NetBeans carregar el projecte i executar.

Execució de Client

En el cas del client el que hem de fer és:

- Obrir una terminal i situar-nos a la carpeta *src*.

- Executar *javac *.java* (Compilar totes les classes)

- Finalment executar el programa amb *java Client <adreça> <port>*

- També el podem executar amb l'opció interactiva per tant passarem un altre argument:

    - 0 per mode Manual: *java Client <adreça> <port> 0*

    - 1 per mode Automàtic: *java Client <adreça> <port> 1*

Conclusions

Aquesta pràctica ha estat molt enriquidora, ja que hem programat un joc mitjançant el RFC que ens van proporcionar i hem après i entès la comunicació via Sockets en Java entre dues o més màquines. També ens ha servit per veure que per petits errors es pot fer que un servidor no respongui o es quedi penjat o deixi de funcionar. Es per això que el Servidor ha de ser la part més robusta d'una aplicació com aquesta. Ha de tenir sempre en compte tots els possibles errors i saber-se recuperar d'ells. No per això també s'ha de tenir un Client que sàpiga per si mateix recuperar-se de possibles errors i també tenir en compte, si es vol fer robust, la verificació de dades que rep, per tal de que el servidor amb el que esta comunicant-se, no tracti d'enganyar a en aquest client.