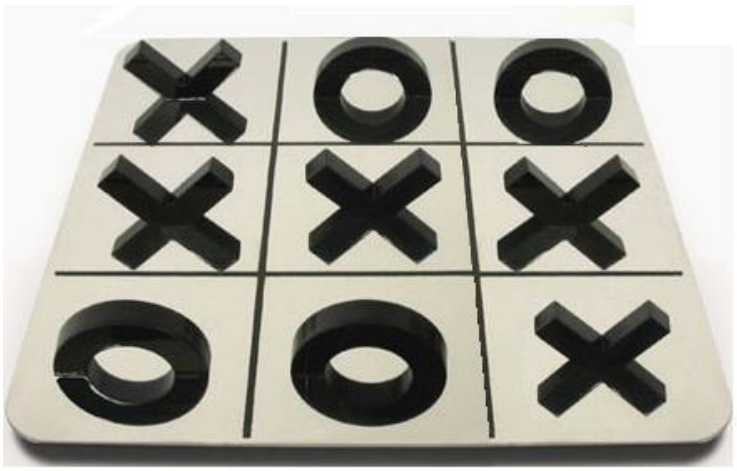
Caso di studio di
Reti di Calcolatori e Internet
Prof. Nicola Di Mauro

a.a. 2008/2009

Titolo progetto: NetTris



Autore

Serafino Francesco

Abstract

NetTris

NetTris è un applicativo Java che consente a più utenti di effettuare partite a Tris in Rete. In particolare l'applicativo è composto da un programma Server che accoglie tutte le richieste dei Client che vogliono effettuare una partita tra loro e le smista secondo la seguente logica:

- Ogni qual volta un nuovo Client desidera effettuare una partita si collega al Server del gioco e attende che il Server trovi un altro Client con cui giocare.

- Se ci sono almeno due Client connessi al Server, quest'ultimo per ogni coppia di Client invia, al primo Client connesso (cronologicamente) il comando di fare da Host e al secondo Client l'IP e la porta del primo Client.

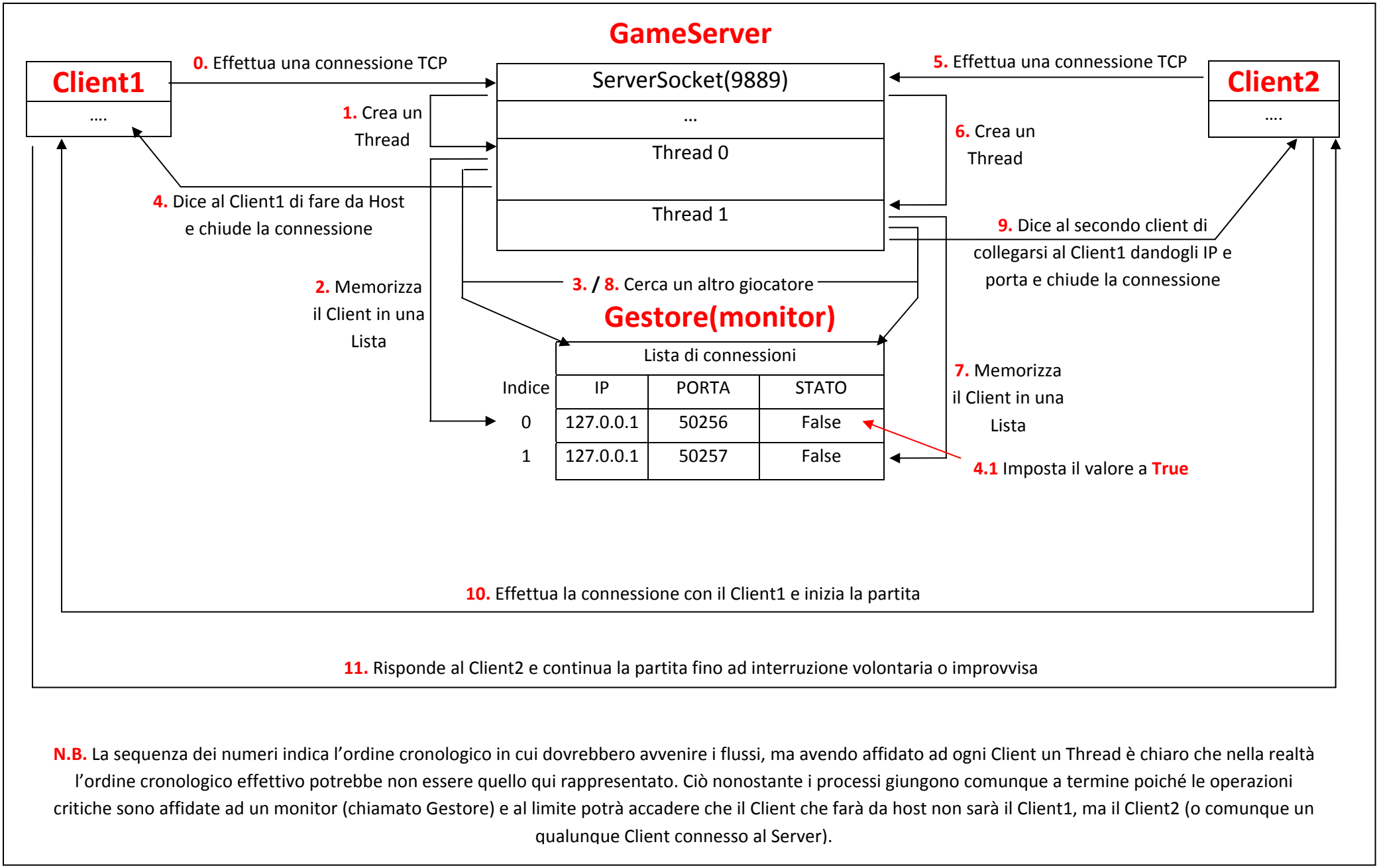
- Se ci sono più Client connessi al Server questi verranno smistati in ordine di arrivo cronologico, ma sempre a due a due. Ciò significa che se il numero dei Client connessi è dispari, ci sarà un giocatore che attenderà fino a quando un nuovo giocatore farà richiesta di giocare o fino a quando un giocatore che è già in partita termina la connessione in cui è impegnato e decide di effettuare un'altra partita.

- Non appena i Client hanno tutte le informazioni per avviare la partita, entrambi chiudono la connessione col server e iniziano a comunicare tra loro. Il primo Client connesso al Server fa da host e l'altro esegue una connessione TCP al primo.

- Ognuno dei due client comunica con l'altro tramite dei flussi di stringhe che alternano un invio di testo ad una ricezione di testo. Questo per far si che ogni volta che un giocatore inserisce la propria mossa questa viene inviata all'altro client e memorizzata da entrambe le parti.

- Alla chiusura della connessione tra i due Client (alla fine della partita o per improvvisa disconnessione della rete) viene posta a entrambi la possibilità di ricollegarsi al server per una nuova partita o, in caso di risposta negativa, l'uscita dal programma.

Schema Grafico



N.B. La sequenza dei numeri indica l'ordine cronologico in cui dovrebbero avvenire i flussi, ma avendo affidato ad ogni Client un Thread è chiaro che nella realtà l'ordine cronologico effettivo potrebbe non essere quello qui rappresentato. Ciò nonostante i processi giungono comunque a termine poiché le operazioni critiche sono affidate ad un monitor (chiamato Gestore) e al limite potrà accadere che il Client che farà da host non sarà il Client1, ma il Client2 (o comunque un qualunque Client connesso al Server).

Progettazione

Informazioni relative alla codifica:

Il programma è stato realizzato in linguaggio Java standard in ambiente JCreator su Windows Vista e testato anche su sistema operativo Ubuntu v8.10.

Il programma è composto da due classi principali, una per implementare il Server e l'altra per implementare il Client. Ogni classe principale possiede delle classi interne (non intese come Inner Class, ma come classi con visibilità di **pacchetto** dichiarate nella classe principale) secondo il seguente schema:

- **GameServer (Server)**
  - Giocatore
  - Gestore
  - Utente
- **NetTris(Client)**
  - Partita

Nei metodi che potrebbero lanciare eccezioni è stata impostata la voce **throws Exception** in modo che se ci dovessero essere eccezioni di qualunque tipo, queste verranno propagate e lanciate, ma sono stati inseriti anche blocchi d'istruzioni **try** – **catch** per catturare e gestire le eccezioni più comuni che si verificano solitamente con la programmazione in rete.

N.B. Con i termini: **Utente**, **Client** e **Giocatore**, si intende sostanzialmente la stessa entità. La differenza del termine è dovuta al contesto particolare in cui la parola è usata, ad esempio se si sta descrivendo il Server in generale è usata la parola Client, se si sta parlando in linea teorica la parola usata è Giocatore, se si sta descrivendo la classe Gestore o la classe Utente la parola usata è Utente.

GameServer

È la classe principale che si occupa di realizzare il server del gioco, e ogni volta che arriva una nuova connessione crea un Thread tramite la classe **Giocatore**.

Metodi:

- **main(String[]): void**
  Metodo principale che avvia il server sulla porta 9889 e rimane in ascolto di nuove connessioni, e per ogni connessione crea un oggetto (Thread) della classe **Giocatore**.

  Variabili e oggetti utilizzati:

Nome	Tipo	Uso
gestore	Gestore	Oggetto Gestore utilizzato per gestire i client connessi
myServer	ServerSocket	Socket di tipo Server per ricevere connessioni in ingresso.
connessione	Socket	Socket collegata al client per tenere libera la ServerSocket e utilizzarla per nuove connessioni in arrivo
giocatore	Giocatore	È un oggetto che estende la classe Thread e rappresenta dal lato Server un client connesso a quest'ultimo

Classi interne (Intese come visibilità di pacchetto):

Giocatore

È la classe che estende la classe Thread e crea un Thread per ogni client connesso al server. Ogni Thread è connesso al proprio client e comunica con esso tramite due Stream di dati con i quali preleva il numero di porta da cui il client sta effettuando la connessione in uscita verso il Server. Inoltre utilizzando la classe **Gestore** preleva i dati per avviare la partita tra due client connessi e li invia al proprio client prima di chiudere la connessione e terminare l'esecuzione.

Variabili d'istanza

Nome	Tipo	Uso
gestore	Gestore	Oggetto Gestore utilizzato per gestire i client connessi
connessione	Socket	Socket collegata al client per tenere libera la ServerSocket e utilizzarla per nuove connessioni in arrivo
inFromClient	BufferedReader	Crea lo stream di input connesso alla Socket connessione
outToClient	DataOutputStream	Crea lo stream di output connesso alla Socket connessione
ip	String	Stringa per memorizzare l'IP del Client
porta	String	Stringa per memorizzare la porta del Client
posizione	int	Valore intero che indica la posizione in una lista di Client connessi (Lista gestita dal Gestore)
utente	Utente	Oggetto Utente che contiene le informazioni del giocatore avversario (Per ogni giocatore connesso)

Costruttori:

- **Giocatore(Socket, Gestore)**
  Costruttore che accetta come parametri (e li memorizza) un oggetto di tipo Socket che è la Socket di connessione col client e un oggetto di tipo gestore che è quello creato nella classe **GameServer** di cui esiste una sola istanza e che fa da **Monitor** per tutti i Thread.

Parametri

Nome	Tipo	Uso
s	Socket	Socket per creare gli Stream di comunicazione e prelevare informazioni quali ip e porta.
g	Gestore	Oggetto Gestore utilizzato per gestire i client connessi

Variabili e oggetti utilizzati:

Nome	Tipo	Uso
gestore	Gestore	Oggetto Gestore utilizzato per gestire i client connessi
connessione	Socket	Socket per creare gli Stream di comunicazione e prelevare informazioni quali ip e porta.
inFromClient	BufferedReader	Crea lo stream di input connesso alla Socket connessione
outToClient	DataOutputStream	Crea lo stream di output connesso alla Socket connessione

Metodi:

- **run(): void**
  Metodo principale che consente di avviare ogni Thread e che si occupa di memorizzare IP e porta del client connesso allo specifico Thread e inviare al Client dei messaggi di risposta all'utente remoto. Inoltre il metodo invia al Gestore i dati relativi al Client e successivamente chiede al gestore se ci sono altri giocatori connessi. In caso affermativo comunica al Client (sempre tramite gli Stream) IP e porta del primo giocatore connesso trovato, altrimenti comunica al Client che, non essendoci altri giocatori, deve fare da Host.

  Variabili e oggetti utilizzati:

Nome	Tipo	Uso
gestore	Gestore	Oggetto Gestore utilizzato per gestire i client connessi
connessione	Socket	Socket collegata al client per tenere libera la ServerSocket e utilizzarla per nuove connessioni in arrivo
inFromClient	BufferedReader	Crea lo stream di input connesso alla Socket connessione
outToClient	DataOutputStream	Crea lo stream di output connesso alla Socket connessione
ip	String	Stringa per memorizzare l'IP del Client
porta	String	Stringa per memorizzare la porta del Client
posizione	int	Valore intero che indica la posizione in una lista di Client connessi (Lista gestita dal Gestore)
utente	Utente	Oggetto Utente che contiene le informazioni del giocatore avversario (Per ogni giocatore connesso)

Gestore

È la classe che svolge la funzione di Monitor per tutti i Thread. Di questa classe viene creata una sola istanza all'inizio e poi passato il riferimento ad ogni Thread come parametro. Questa classe possiede due variabili di istanza e sono due ArrayList: quello principale che contiene oggetti di tipo Utente che a loro volta contengono le informazioni di ogni Client e un secondo ArrayList di interi che contiene le posizioni riutilizzabili nel primo ArrayList. Una posizione è intesa come riutilizzabile se l'utente al suo interno memorizzato non è più utile poiché è stato già messo in comunicazione con un altro utente.

Variabili d'istanza

Nome	Tipo	Uso
utenti	ArrayList<Utente>	ArrayList che contiene tutti gli Utenti connessi in attesa di giocare e utenti riutilizzabili
posti	ArrayList<int>	ArrayList di interi che contiene tutte le posizioni riutilizzabili del precedente ArrayList

Costruttori:

- **Gestore()**
  La classe possiede solo un costruttore che non accetta parametri ed ha corpo vuoto.

Metodi:

- **synchronized entra(String, String): int**
  Metodo sincronizzato (Monitor) al quale fanno accesso i Thread della classe **Giocatore** per inserire i dati del Client all'interno dell'ArrayList utenti, cioè IP e porta. In più questo metodo oltre a memorizzare l'IP e la Porta del Client assegna anche una variabile booleana per memorizzare lo stato del Client e la imposta a False. (False: il Client è appena entrato oppure ha trovato un altro giocatore, True: non è stato trovato nessun giocatore e il Client si imposta a true e inizia a fare da host)

  Parametri

Nome	Tipo	Uso
ip	String	Stringa per memorizzare l'IP del Client
porta	String	Stringa per memorizzare la porta del Client

  Variabili e oggetti utilizzati:

Nome	Tipo	Uso
posizione	int	Valore intero che indica la posizione in una lista di Client connessi (Lista gestita dal Gestore)
utenti	ArrayList<Utente>	ArrayList che contiene tutti gli Utenti connessi in attesa di giocare e utenti riutilizzabili
posti	ArrayList<int>	ArrayList di interi che contiene tutte le posizioni riutilizzabili del precedente ArrayList

- **synchronized avviaPartita(int): Utente**
  Metodo sincronizzato (Monitor) al quale fanno accesso i Thread della classe **Giocatore** per cercare un altro giocatore con cui iniziare la partita. La ricerca si basa sulla scansione dell'ArrayList e se si trova un Utente con stato true si dice al client di contattare il giocatore con stato true tramite IP e porta. Se la ricerca non porta a risultati significa che nessuno è connesso e quindi il client dovrà lui per primo avviare una partita e quindi fare da host e attendere che qualche altro giocatore lo contatti.

  Parametri

Nome	Tipo	Uso
posizione	int	Posizione all'interno dell'ArrayList del client che effettua la richiesta di avvio partita

  Variabili e oggetti utilizzati:

Nome	Tipo	Uso
utenti	ArrayList<Utente>	ArrayList che contiene tutti gli Utenti connessi in attesa di giocare e utenti riutilizzabili
posti	ArrayList<int>	ArrayList di interi che contiene tutte le posizioni riutilizzabili del precedente ArrayList
temp	Utente	Oggetto che contiene temporaneamente i dati relativi al giocatore avversario da contattare o i propri se non c'è nessuno collegato
i	int	Contatore per eseguire un ciclo
trovato	boolean	Variabile booleana per uscire dal ciclo prima della fine
miaPosizione	int	Posizione del client all'interno dell'ArrayList
altraPosizione	int	Posizione dell'avversario all'interno dell'ArrayList

Utente

È la classe che rappresenta il client durante tutta l'applicazione Server. E' composta esclusivamente da tre variabili d'istanza che corrispondono all'IP, alla porta e allo stato. Inoltre al suo interno sono previsti tre metodi di accesso che restituiscono rispettivamente i tre dati poco sopra descritti (IP, porta e stato).

Variabili d'istanza

Nome	Tipo	Uso
indirizzo	String	Stringa per memorizzare l'IP del Client
porta	String	Stringa per memorizzare la porta del Client
stato	boolean	Stringa per memorizzare lo stato del Client

Costruttori:

- **Utente()**
  Costruttore che non accetta parametri ed ha corpo vuoto.

- **Utente(String, String, boolean)**
  Costruttore che prende come parametri due stringhe relative all'IP e la porta e una variabile booleana per lo stato e li memorizza all'interno delle variabili d'istanza.

  Parametri

Nome	Tipo	Uso
ip	String	Stringa con l'IP del Client
port	String	Stringa con la porta del Client
stat	boolean	Stringa con lo stato del Client

  Variabili e oggetti utilizzati:

Nome	Tipo	Uso
indirizzo	String	Stringa per memorizzare l'IP del Client
porta	String	Stringa per memorizzare la porta del Client
stato	boolean	Stringa per memorizzare lo stato del Client

Metodi:

- **getIp(): String**
  Metodo di accesso che restituisce l'IP al chiamante

  Variabili e oggetti utilizzati:

Nome	Tipo	Uso
indirizzo	String	Stringa che contiene l'IP del Client

- **getPort(): String**
  Metodo di accesso che restituisce la porta al chiamante

  Variabili e oggetti utilizzati:

Nome	Tipo	Uso
porta	String	Stringa che contiene la porta del Client

- **getStat(): String**
  Metodo di accesso che restituisce lo stato al chiamante

  Variabili e oggetti utilizzati:

Nome	Tipo	Uso
stato	boolean	Stringa che contiene lo stato del Client

NetTris

È la classe che si collega al server da cui reperisce tutte le informazioni relative ad un altro giocatore connesso e avvia la partita con quest'ultimo facendo da Server o da Client a seconda dai casi. Utilizza la classe Partita per dare inizio al gioco vero e proprio.

Metodi:

- **main(String[]): void**
  Metodo principale che consente di collegarsi al **GameServer** e di ricevere le istruzioni per la connessione tra client. Inoltre questo metodo avvia un ciclo che dura per l'intera esecuzione del programma per consentire all'utente di uscire quando lo desidera, oppure di effettuare quante partite desidera, o di vedere le istruzioni ogni volta che lo desidera. Contiene al suo interno una serie di blocchi di scelta nidificati per effettuare il sottoprogramma specifico. Le istruzioni sono stampate a video tramite una sequenza di istruzioni di stampa sull'output standard.

  Variabili e oggetti utilizzati:

Nome	Tipo	Uso
inFromUser	BufferedReader	Buffer collegato all'input standard (tastiera)
input	String	Variabile che contiene le istruzioni prese da tastiera dall'utente
prova	InetAddress	Contiene l'IP del server, inserito dall'utente. Se l'utente non digita nulla verrà preso il localhost
client	Socket	Socket per la connessione al GameServer
inFromServer	BufferedReader	Stream per la ricezione dei dati dal Server
outToServer	DataOutputStream	Stream per l'invio dei dati al Server
porta	int	Porta da cui esce la connessione verso il Server
portaString	String	È la porta sopra descritta però convertita in stringa
ipHost	String	Stringa che contiene le istruzioni per la connessione (la scritta "host" o l'IP dell'avversario)
portaHostString	String	Stringa che contiene la porta con la quale si è effettuata la connessione al GameServer o la porta dell'avversario
portaHost	int	È la porta appena descritta però convertita in intero
partita	Partita	Oggetto di tipo Partita che crea ed avvia la partita vera e propria

Classi interne (Intese come visibilità di pacchetto):

Partita

È la classe che consente realmente a i due client di giocare tra loro. I suoi metodi e costruttori sono specifici per realizzare le regole del gioco del tris, sviluppate appositamente per un gioco che avviene tra due giocatori reali connessi tra loro.

Variabili d'istanza

Nome	Tipo	Uso
table	String[ ][ ]	Matrice di Stringhe 3x3 per memorizzare la tabella di gioco del Tris

Costruttori:

- **Partita()**
  La classe possiede solo un costruttore che non accetta parametri ed ha corpo vuoto.

Metodi:

- **host(int): void**
  Metodo che avvia la comunicazione "alla pari" tra il client e l'avversario, impostando il client come host. Ciò significa che verrà eseguito questo metodo se il giocatore è il primo ad essere connesso e quindi è in attesa che un altro giocatore si colleghi. Questo metodo effettua un ciclo per un massimo cinque volte (cinque è il numero massimo di mosse che un giocatore può fare in una partita a Tris) il cui corpo è composto da una ricezione della mossa effettuata dall'altro giocatore e, a seguire, un invio della propria mossa all'avversario.
  Il ciclo si interrompe quando uno dei due giocatori vince oppure quando uno dei due giocatori decide di terminare prematuramente l'applicazione, altrimenti verrà eseguito per intero e a quel punto la partita sarà patta. Ad ogni mossa effettuata (sia inviata che ricevuta) verrà effettuato un controllo per verificare se uno dei due giocatori ha vinto.

  Parametri

Nome	Tipo	Uso
porta	int	È la porta con la quele il client si era connesso al GameServer e adesso verrà riciclata per creare un server su di essa

  Variabili e oggetti utilizzati:

Nome	Tipo	Uso
myServer	ServerSocket	Stringa per memorizzare l'IP del Client
mossaUscita	String	Stringa per memorizzare la mossa fatta dall'utente e da inviare all'altro client
mossaEntrata	String	Stringa per memorizzare la mossa fatta dall'altro client e da inserire all'interno della matrice
r	String	Contiene il carattere che indica la riga della matrice in cui posizionare la mossa (estratto dalla mossaUscita o mossaEntrata)
c	String	Contiene il carattere che indica la colonna della matrice in cui posizionare la mossa (estratto dalla mossaUscita o mossaEntrata)
riga	int	È la conversione in intero della Stringa r
colonna	int	È la conversione in intero della Stringa r
connessione	Socket	È la Socket che si crea quando l'altro client effettua la connessione
inFromUser	BufferedReader	Stream di dati collegato all'input standard (la tastiera)
inFromPeer	BufferedReader	Stream di dati collegato alla Socket e utilizzato per ricevere la mossa dall'altro client
outToPeer	DataOutputStream	Stream di dati collegato alla Socket e utilizzato per inviare la propria mossa all'altro client
i	int	Contatore standard per eseguire cicli for
table	String[ ][ ]	Matrice di Stringhe 3x3 per memorizzare la tabella di gioco del Tris

- **join(String, int): void**
  Metodo simmetrico al metodo **host**. Avvia la comunicazione "alla pari" tra il client e l'avversario, facendo connettere il client ad un altro client che in questo momento sta facendo da server. Ciò significa che verrà eseguito questo metodo se il giocatore è il secondo ad essere connesso e quindi ha trovato un altro giocatore collegato. Questo metodo effettua un ciclo per un massimo cinque volte (cinque è il numero massimo di mosse che un giocatore può fare in una partita a Tris) il cui corpo è composto da un invio della propria mossa all'avversario e, a seguire, una ricezione della mossa effettuata dall'altro giocatore.

Il ciclo si interrompe quando uno dei due giocatori vince oppure quando uno dei due giocatori decide di terminare prematuramente l'applicazione, altrimenti verrà eseguito per intero e a quel punto la partita sarà patta. Ad ogni mossa effettuata (sia inviata che ricevuta) verrà effettuato un controllo per verificare se uno dei due giocatori ha vinto.

Parametri

Nome	Tipo	Uso
ip	String	Contiene l'IP dell'avversario al quale bisogna effettuare una connessione TCP.
porta	int	Contiene la porta sulla quale l'avversario sta facendo hosting. Necessaria insieme all'IP per stabilire la connessione.

Variabili e oggetti utilizzati:

Nome	Tipo	Uso
mossaUscita	String	Stringa per memorizzare la mossa fatta dall'utente e da inviare all'altro client
mossaEntrata	String	Stringa per memorizzare la mossa fatta dall'altro client e da inserire all'interno della matrice
r	String	Contiene il carattere che indica la riga della matrice in cui posizionare la mossa (estratto dalla mossaUscita o mossaEntrata)
c	String	Contiene il carattere che indica la colonna della matrice in cui posizionare la mossa (estratto dalla mossaUscita o mossaEntrata)
riga	int	È la conversione in intero della Stringa r
colonna	int	È la conversione in intero della Stringa r
connessione	Socket	È la Socket che si crea quando si effettua la connessione TCP all'altro client
inFromUser	BufferedReader	Stream di dati collegato all'input standard (la tastiera)
inFromPeer	BufferedReader	Stream di dati collegato alla Socket e utilizzato per ricevere la mossa dall'altro client
outToPeer	DataOutputStream	Stream di dati collegato alla Socket e utilizzato per inviare la propria mossa all'altro client
i	int	Contatore standard per eseguire cicli for
table	String[ ][ ]	Matrice di Stringhe 3x3 per memorizzare la tabella di gioco del Tris

- **printTable(): void**
  Metodo che stampa a video lo stato attuale della matrice utilizzando delle istruzioni di stampa ripetute e utilizzando i caratteri "+, -, |" per costruire graficamente una tabella.

Variabili e oggetti utilizzati:

Nome	Tipo	Uso
table	String[ ][ ]	Matrice di Stringhe 3x3 per memorizzare la tabella di gioco del Tris

- **setToEmpty(): void**
  All'avvio della partita inizializza la matrice di stringhe inserendo in ogni posizione il carattere " " (spazio e non stringa vuota), in modo tale che appaia vuota.

Variabili e oggetti utilizzati:

Nome	Tipo	Uso
i	int	Contatore per scansionare la matrice (per le righe)
j	int	Contatore per scansionare la matrice (per le colonne)
table	String[ ][ ]	Matrice di Stringhe 3x3 per memorizzare la tabella di gioco del Tris

- **haveWin(String): String**
  Metodo che controlla se sulla diagonale principale, sulla diagonale secondaria, su tutte le righe e su tutte le colonne ci sono tre simboli uguali (ma diversi dal carattere di spazio). In caso affermativo restituisce il simbolo trovato (X o O) e la partita termina, altrimenti restituisce stringa vuota ("") e si procede con l'esecuzione del programma. Richiede come parametro il simbolo del giocatore chiamante in modo da stampare a video in maniera corretta i messaggi per l'utente.

Parametri

Nome	Tipo	Uso
giocatore	String	Stringa che contiene il simbolo del giocatore (X o O)

Variabili e oggetti utilizzati:

Nome	Tipo	Uso
trovato	boolean	Si imposta a true se durante l'esecuzione del metodo si trova subito una riga o colonna o diagonale con simboli uguali ed evita il controllo fino alla fine
i	int	Contatore per i cicli for (per esaminare le righe e poi le colonne)
table	String[ ][ ]	Matrice di Stringhe 3x3 per memorizzare la tabella di gioco del Tris

- **cell(String): String**
  Metodo che consente di trasformare il carattere inserito da tastiera che indica la posizione in cui si vuole inserire la propria mossa nelle coordinate reali della matrice. Questo perché per poter giocare gli utenti indicano la propria mossa digitando il tasto corrispondente ai primi tre tasti delle tre righe della tastiera (QWE – ASD - ZXC), in questo modo è facile anche visivamente inserire mosse intuitive e corrette.

Parametri

Nome	Tipo	Uso
comando	String	Contiene una delle lettere consentite (QWE – ASD – ZXC)

Variabili e oggetti utilizzati:

Nome	Tipo	Uso
--	--	--

- **isEmpty(int, int): boolean**
  Metodo che controlla, prima di inserire una mossa, se la casella in cui si desidera inserire la propria mossa è vuota. In caso affermativo allora si procederà con l'inserimento della mossa, altrimenti si chiederà all'utente di inserirne una valida. Accetta come parametro le coordinate della casella e restituisce true se la casella è libera, altrimenti false.

Parametri

Nome	Tipo	Uso
r	int	Coordinata della riga
c	int	Coordinata della colonna

Variabili e oggetti utilizzati:

Nome	Tipo	Uso
table	String[ ][ ]	Matrice di Stringhe 3x3 per memorizzare la tabella di gioco del Tris