

# Sette e Mezzo

Documentazione del caso di studio



**METODI AVANZATI DI PROGRAMMAZIONE 2008/2009**  
**Corso di Laurea: Informatica e Tecnologie per la produzione del software**

Prof. Michelangelo Ceci  
Autori:  
Tarantino Maria Cristina - Turturo Fabio

# Sette e Mezzo

## Indice

### Indice

1.	INTRODUZIONE.....	4
1.1.	REGOLE DEL GIOCO.....	5
2.	CARATTERISTICHE DEL SISTEMA .....	6
3.	STAKEHOLDERS.....	8
4.	PRODOTTO .....	9
4.1.	SETTE E MEZZO – STAND ALONE .....	9
4.2.	SETTE E MEZZO – CLIENT/SERVER .....	9
4.3.	PARTI COMUNI.....	9
4.4.	PARTI VARIANTI.....	9
4.5.	MODELLO DI DECISIONE.....	10
5.	REQUISITI OPERATIVI.....	11
5.1.	REQUISITI SOFTWARE.....	11
5.2.	REQUISITI HARDWARE.....	11
6.	PROGETTAZIONE DEL SISTEMA.....	12
6.1.	MODELLO DI DECISIONE.....	15
6.2.	SPECIFICA DEI PACKAGE E DELLE CLASSI .....	16
6.2.1.	SETTEMEZZO.GIOCATORE.....	16
6.2.1.1.	GIOCATORE .....	17
6.2.1.2.	GIOCATOREUMANO.....	20
6.2.1.3.	BANCOCPU.....	20
6.2.1.4.	SFIDANTECPU .....	20
6.2.2.	SETTEMEZZO.MAZZO.....	21
6.2.2.1.	MAZZO .....	22
6.2.2.2.	MAZZONAPOLETANO.....	26
6.2.2.3.	CARTA.....	26
6.2.2.4.	SEME.....	26
6.2.2.5.	TIPOCARTA.....	26
6.2.3.	SETTEMEZZO.UTIL.EXCEPTION.....	27
6.2.3.1.	MAZZOTERMINATOEXCEPTION.....	27
6.2.3.2.	SBALLATOEXCEPTION.....	27
6.2.3.3.	SETTEMEZZOEXCEPTION.....	27
6.2.4.	SETTEMEZZO.UTIL.DATABASE.....	28
6.2.4.1.	ALIAS .....	29
6.2.4.2.	DBACCESS.....	30
6.2.4.3.	IPARTITADAO.....	31
6.2.4.4.	PARTITADAOFACTORY .....	32
6.2.4.5.	PARTITAEENTRY .....	32
6.2.4.6.	TRANSAZIONEDB.....	32
6.2.4.7.	UTENTEENTRY .....	32
6.2.5.	SETTEMEZZO.....	33
6.2.5.1.	CLASSEMAIN .....	33
6.2.5.2.	PARTITA.....	34
6.2.6.	SETTEMEZZO.INTERFACCIAUTENTE .....	36

6.2.6.1.	ICONSOLE.....	38
6.2.6.2.	CLI.....	38
6.2.6.3.	GUI.....	38
6.2.7.	SETTEMEZZO.INTERFACCIAUTENTE.UTIL.....	38
6.2.7.1.	ICONDITION.....	39
6.2.7.2.	DEFAULTCONDITION.....	40
6.2.7.3.	KEYBOARD.....	40
6.2.8.	SETTEMEZZO.INTERFACCIAUTENTE.COMPONENTI.....	41
6.2.8.1.	INPUTDIALOG.....	42
6.2.8.2.	TABELLASTORICO.....	43
6.2.9.	SETTEMEZZO.INTERFACCIAUTENTE.COMPONENTI.CALENDARIO.....	44
6.2.9.1.	CALENDARIO.....	44
6.2.9.2.	ANNO.....	45
6.2.9.3.	MESE.....	45
6.2.9.4.	GIORNO.....	45
6.2.10.	SETTEMEZZO.SERVER.....	46
6.2.10.1.	MULTISERVER.....	46
6.2.10.2.	SERVEONECLIENT.....	47
6.2.10.3.	PARTITA.....	49
6.2.11.	SETTEMEZZO.CLIENT.....	50
6.2.11.1.	GIOCATORECLIENT.....	51
6.2.12.	SETTEMEZZO.APPLLET.....	52
6.2.12.1.	MAINAPPLLET.....	52
6.2.12.2.	MEDIATORE.....	53
6.2.13.	SETTEMEZZO.APPLLET.UTIL.....	54
6.2.14.	SETTEMEZZO.APPLLET.COMPONENTI.....	54
6.2.15.	SETTEMEZZO.APPLLET.COMPONENTI.HELP.....	55
6.2.15.1.	HELPPFRAME.....	55
6.2.15.2.	INODE.....	56
6.2.15.3.	NODO.....	56
6.3.	MODELLO DI DECISIONE.....	57
6.4.	PROGETTAZIONE DATABASE.....	59
6.4.1.	DETTAGLIO DEI DATI.....	59
6.4.2.	MODELLO DEL DATABASE.....	61
6.4.3.	SCRIPT SQL.....	62
6.5.	PAGINE JSP.....	64
6.5.1.	SAVEUSER.JSP.....	64
6.5.2.	STORICO.JSP.....	65
7.	GUIDE.....	66
7.1.	SVOLGIMENTO DEL GIOCO.....	66
7.2.	GUIDA VERSIONE STAND ALONE.....	67
7.2.1.	INSTALLAZIONE SU SISTEMA OPERATIVO WINDOWS.....	67
7.2.2.	INSTALLAZIONE SU SISTEMA OPERATIVO LINUX.....	67
7.2.3.	AVVIARE SETTE E MEZZO.....	68
7.2.4.	QUANDO SI AVVIA SETTE E MEZZO.....	69
7.2.5.	SALVARE UNA PARTITA.....	73
7.2.6.	CARICARE UNA PARTITA.....	74
7.3.	GUIDA VERSIONE CLIENT/SERVER LATO CLIENT.....	75
7.3.1.	AVVIARE SETTE E MEZZO.....	75
7.3.2.	BARRA DEI MENÙ.....	76

7.3.3.	AREA DI GIOCO .....	77
7.3.4.	COME REGISTRARSI .....	78
7.3.5.	COME AUTENTICARSI .....	80
7.3.6.	SALVARE UNA PARTITA.....	80
7.3.7.	CARICARE UNA PARTITA .....	80
7.3.8.	STORICO PERSONALE DELLE PARTITE .....	81
7.4.	GUIDA ALL'INSTALLAZIONE DELL'APPLICAZIONE CLIENT/SERVER LATO SERVER .....	82
8.	ESEMPI DI TEST.....	83
8.1.	ENTRAMBI I GIOCATORI REALIZZANO LO STESSO PUNTEGGIO: VINCE IL BANCO.....	83
8.2.	LO SFIDANTE SBALLA: VINCE IL BANCO.....	84
8.3.	IL BANCO SBALLA: VINCE LO SFIDANTE.....	85
8.4.	LO SFIDANTE TOTALIZZA UN PUNTEGGIO SUPERIORE AL BANCO: VINCE LO SFIDANTE.....	86
8.5.	LO SFIDANTE TOTALIZZA 7 ½ CON MATT A E FIGURA: VINCE LO SFIDANTE CHE DIVENTA BANCO. IL MAZZO VIENE MESCOLATO.....	87
8.6.	LO SFIDANTE TOTALIZZA 7 ½ E IL BANCO SBALLA: VINCE LO SFIDANTE.....	90
8.7.	LO SFIDANTE TOTALIZZA 7 ½ E IL BANCO TOTALIZZA UN PUNTEGGIO INFERIORE A 7 ½: VINCE LO SFIDANTE E DIVENTA BANCO.....	91
8.8.	ERRORI DI COMPILAZIONE FORM.....	92
8.9.	INDIRIZZO E-MAIL GIÀ PRESENTE NEL DATABASE DEL SISTEMA.....	94
8.10.	ERRORI DI ACCESSO AL SISTEMA.....	95
9.	SVILUPPI FUTURI.....	96
10.	GLOSSARIO.....	97

# Sette e Mezzo

---

## Introduzione

### 1. Introduzione

Sette e Mezzo è uno dei più diffusi giochi d'azzardo italiani, tradizionalmente giocato nelle riunioni familiari del periodo natalizio. È ricco di tattiche e astuzie di gioco e ha molte varianti.

L'applicazione "Sette e Mezzo" implementa una variante del famoso gioco italiano del sette e mezzo. Essa è stata sviluppata in linguaggio Java (SDK versione 1.6).

La scelta di un linguaggio di programmazione che supporti il paradigma *object oriented* deriva dall'esigenza di garantire:

- un possibile riuso delle componenti software;
- la semplificazione del processo di porting del software su piattaforme differenti. Quest'ultimo punto è stato reso possibile isolando specifiche funzionalità o servizi all'interno di appositi package o classi progettate appositamente, in modo tale da poter modificare singole componenti del progetto, senza doversi preoccupare dell'impatto che tali modifiche possono avere sulle restanti componenti software.

### 1.1. Regole del gioco

Lo scopo del gioco è quello di realizzare il punteggio più alto possibile senza mai sballare, vale a dire senza superare il sette e mezzo.

Il mazziere deve cercare di eguagliare o superare il punteggio del giocatore avversario senza sballare. Il banco:

- riscuote la somma puntata dallo sfidante se questi ha sballato, o ha totalizzato un punteggio inferiore o uguale al suo.
- paga l'equivalente della puntata del suo avversario se ha superato il punteggio di quest'ultimo.

Il punteggio di ciascun giocatore si calcola sommando i punti di tutte le carte che possiede.

- Le carte dall'asso al sette valgono tanti punti quanto è il loro valore numerico. L'asso vale un punto, il due vale due punti, ecc.
- Le figure valgono mezzo punto.
- Il re di denari ha la funzione di MATTA: può assumere il punteggio di una qualsiasi altra carta a discrezione del giocatore che la possiede. Può, dunque, assumere qualunque punteggio intero compreso tra 1 e 7, oppure punteggio 0.5.

Il sette e mezzo realizzato con due sole carte (un sette e una figura, oppure la matta e una figura oppure la matta e un sette) è detto sette e mezzo reale o sette e mezzo legittimo o anche sette e mezzo d'emblée. Quando il giocatore realizza questo punteggio riceve dal banco una somma pari al doppio della posta e diventa mazziere alla mano successiva.

Se è il mazziere a fare sette e mezzo reale, egli riscuote una posta doppia dal giocatore; nel caso in cui quest'ultimo abbia fatto a sua volta sette e mezzo reale o abbia sballato, riscuote una posta semplice, e non deve cedere il banco.

Se il giocatore realizza il sette e mezzo reale avendo la Matta e una figura (es. re di denari ed una figura qualunque), il mazziere vince solo se realizza a sua volta un sette e mezzo reale usando solo due carte, quindi un sette e una figura.

Se il mazziere realizza un sette e mezzo utilizzando più di due carte si dice che è illegittimo e può battere il suo avversario a meno che quest'ultimo non realizzi un sette e mezzo con la matta e figura.

## 2. Caratteristiche del sistema

Andiamo a presentare le caratteristiche del sistema, richieste dal docente. Il sistema deve:

- C1. modellare il gioco del Sette e Mezzo in modo da rappresentare la carta da gioco e il mazzo di carte napoletane;
- C2. essere dotato di interfaccia grafica: il sistema deve consentire all'utente d'interagire con l'applicazione mediante finestre, icone, menu e dispositivo di puntamento;
- C3. essere dotato di un'interfaccia a riga di comando: il sistema deve consentire all'utente d'interagire con l'applicazione inviando comandi tramite tastiera e ricevendo risposte alle elaborazioni tramite testo scritto;
- C4. essere utilizzabile mediante la rete internet: il sistema deve essere accessibile mediante il semplice utilizzo di un browser abbinato ad una rete internet. Questo è possibile attraverso l'implementazione di un applet;
- C5. permettere di visualizzare lo storico delle partite effettuate: il sistema deve implementare una base di dati per tener traccia delle partite giocate e salvate, per ogni mano, i punteggi di ogni giocatore, il suo nome e l'esito della mano;
- C6. permettere il salvataggio e il caricamento di una determinata partita.

Mostriamo di seguito le capacità implementate nell'applicazione in esame.

- C1. modellare il gioco del Sette e Mezzo in modo da rappresentare la carta da gioco e il mazzo di carte napoletane;
- C2. essere dotato di interfaccia grafica;
- C3. essere dotato di un'interfaccia a riga di comando;
- C4. essere utilizzabile mediante la rete internet;
- C5. permettere di visualizzare lo storico delle partite effettuate;
- C6. permettere il salvataggio e il caricamento di una determinata partita;
- C7. gestire gli eventuali errori causati dall'utente: all'atto di un inserimento, il sistema controlla la possibilità di eseguire tale operazione e la consistenza dei dati immessi. Questo permette al sistema di continuare la sua esecuzione nonostante il riscontro di errori causati dall'utente.
- C8. prevedere un sistema di messaggi di notifica: all'atto del salvataggio e del caricamento di una partita il sistema notifica l'esito dell'operazione. Inoltre, durante l'esecuzione del gioco, l'utente deve essere aggiornato costantemente sullo stato della partita;
- C9. prevedere un sistema di registrazione: il programma deve prevedere un sistema di registrazione per accedere ai servizi aggiuntivi. Questo meccanismo deve essere implementato attraverso la tecnologia [JSP](#);
- C10. prevedere un sistema di [autenticazione](#): il programma deve prevedere l'autenticazione dell'utente registrato attraverso nome utente e password. Questo permette all'utente di usufruire di servizi aggiuntivi;
- C11. essere utilizzabile localmente.

La tabella seguente evidenzia l'estensione del nostro progetto rispetto a quella del docente.

Caratteristiche del sistema	Progetto docente	Progetto sviluppato
C1	✓	✓
C2	✓	✓
C3	✓	✓
C4	✓	✓
C5	✓	✓
C6	✓	✓
C7	✓	✓
C8		✓
C9		✓
C10		✓
C11		✓



### 3. Stakeholders

L'applicazione è rivolta a:

#### **COMMITTENTE**

Determina le caratteristiche minime che il sistema deve fornire.

#### **GIOCATORE**

Interessato alle seguenti funzioni offerte dal sistema:

- possibilità di giocare;
- possibilità di visionare lo storico delle partite effettuate per una data che egli stesso sceglie;
- possibilità di salvare lo stato della partita alla fine di ogni mano;
- possibilità di caricare una partita precedentemente salvata.

#### **AMMINISTRATORE DEL SERVER**

Fa uso del sistema lato server per fornire un servizio ludico aggiuntivo nel suo portale web.

#### **SVILUPPATORI**

Interessati allo sviluppo del progetto, all'interazione con il database tramite linguaggio di programmazione, all'interazione dell'applicazione lato client e server ed a tutti i possibili metodi di integrazione utilizzati nel sistema.

## 4. Prodotto

Durante lo sviluppo del sistema, si è ritenuto opportuno distinguere due diversi prodotti: Stand Alone, che evidenziasse l'implementazione del sistema ai primordi dello sviluppo, e Client/Server, che implementasse un'estensione delle caratteristiche minime richieste dal docente. L'obiettivo di questa strategia è mostrare come si è evoluto la nostra progettazione del sistema.

### 4.1. Sette e Mezzo – Stand Alone

Il prodotto consente di giocare contro la CPU ad una partita a Sette e Mezzo su un singolo computer. La partita può essere eseguita sia in modalità grafica che in modalità testuale, a seconda delle preferenze dell'utente.

### 4.2. Sette e Mezzo – Client/Server

Il prodotto consente all'utente di giocare da un qualsiasi computer che presenti un accesso internet, un [browser](#) e una Java Virtual Machine: ciò è possibile grazie ad un applet, particolare tipo di programma Java il cui output è pensato per essere incluso in pagine Web. Quando un utente visualizza la pagina Web che contiene l'applet, questa viene eseguita localmente (cioè sulla macchina dell'utente) e non sul server. Inoltre il software consente a più giocatori di connettersi contemporaneamente al server in modo da giocare una partita. Permette, solo agli utenti registrati, l'autenticazione in modo tale da usufruire delle funzionalità di caricamento e salvataggio delle partite, e visualizzazione dello storico delle partite personali.

### 4.3. Parti comuni

- C1. modellare il gioco del Sette e Mezzo in modo da rappresentare la carta da gioco e il mazzo di carte napoletane;
- C2. essere dotato di interfaccia grafica;
- C5. permettere di visualizzare lo storico delle partite effettuate;
- C6. permettere il salvataggio e il caricamento di una determinata partita;
- C7. gestire gli eventuali errori causati dall'utente;
- C8. prevedere un sistema di messaggi di notifica.

### 4.4. Parti varianti

- C3. essere dotato di un'interfaccia a riga di comando;
- C4. essere utilizzabile mediante la rete internet;
- C9. prevedere un sistema di registrazione;
- C10. prevedere un sistema di [autenticazione](#);
- C11. essere utilizzabile localmente.

#### 4.5. Modello di decisione

La tabella riassume le caratteristiche del sistema differenziate rispetto ai prodotti appena delineati.

	Capacità	Prodotti	
		Stand Alone	Client / Server
Comuni	C1	✓	✓
	C2	✓	✓
	C5	✓	✓
	C6	✓	✓
	C7	✓	✓
	C8	✓	✓
Varianti	C3	✓	
	C4		✓
	C9		✓
	C10		✓
	C11	✓	

## 5. Requisiti operativi

In questa sezione saranno presentati i requisiti, sia hardware che software, necessari al funzionamento dell'applicazione in esame.

### 5.1. Requisiti Software

#### Versione Stand Alone

- [Sistema operativo](#) Windows Xp, Vista o superiore, Linux a 32 o 64 bit;
- [Ambiente di esecuzione di programmi java JRE](#) versione 6;
- [Database relazionale](#) MySQL versione 5.0;
- [Web server](#) Apache Tomcat versione 6.

#### Versione Client/Server

##### Lato Client

- [Browser web](#):
  - Mozilla Firefox versione 2 o superiore;
  - Internet Explorer versione 6 o superiore;
- Ambiente di esecuzione di programmi java JRE versione 6.

##### Lato Server

- [Sistema operativo](#) Windows Xp, Vista o superiore, Linux a 32 o 64 bit;
- [Ambiente di esecuzione di programmi java JRE](#) versione 6;
- [Database relazionale](#) MySQL versione 5.0;
- [Web server](#) Apache Tomcat versione 6.

### 5.2. Requisiti Hardware

- computer con velocità di clock del processore da 300 MHz o superiore; processore Intel Quad core/Dual core/Pentium/Celeron, AMD K6/Athlon/Duron o compatibile.
- 512 MB di RAM o superiore;
- 1,5 GB di spazio disponibile su disco rigido;
- scheda video e monitor con risoluzione 800 x 600 o superiore;
- connessione internet richiesta per versione Client/Server;
- Tastiera e mouse.

# Sette e Mezzo

## Progettazione

### 6. Progettazione del sistema

In questa sezione andremo ad analizzare il progetto di entrambi i prodotti in maniera top-down: per ciascun package si descriverà la specifica dettagliando le classi che lo compongono. Saranno inoltre presentate, man mano che si incontreranno, le differenze che intercorrono tra la versione Stand Alone e quella Client/Server del software e si evidenzieranno le modifiche che sono state apportate rispetto al progetto presentato a lezione.

Abbiamo cercato di rispettare il più possibile la specifica dei package fornitaci a lezione. Le modifiche ad essa apportate sono:

- aggiunta di sottopackage e classi al package “*interfacciautente*” per migliorare sia la strutturazione delle classi, sia l’interazione utente – sistema;
- introduzione del package “*settemezzo.util.exception*” per raggruppare le eccezioni personalizzate;
- revisione del package relativo all’interazione con il database, al fine di rendere il programma il programma facilmente espandibile nel caso si ritenga necessario interagire con DBMS differenti da quello utilizzato;

Inoltre nel progetto lato client è stato ridotto il numero di classi in esso contenute, rispetto al progetto presentato a lezione, per consentire una interazione più rapida e realizzare un sistema più facilmente mantenibile e riusabile.

Le modifiche rispetto alla specifica di base, non si presentano solo a livello dei package, ma anche a livello delle classi. Rispetto al progetto proposto, si nota l’esclusione del campo *iConsole* dalla classe *Giocatore* in modo da avere una separazione tra i due livelli. Inoltre si è aggiunta una struttura che tiene traccia delle carte in possesso di ciascun giocatore.

Le modifiche riguardano anche la parte relativa al mazzo di carte: una classe astratta *Mazzo* è stata creata per consentire la creazione di mazzi diversi da quello napoletano. La classe *Carta* si è arricchita di un campo che consente di facilitare l’individuazione delle carte estratte da un mazzo.

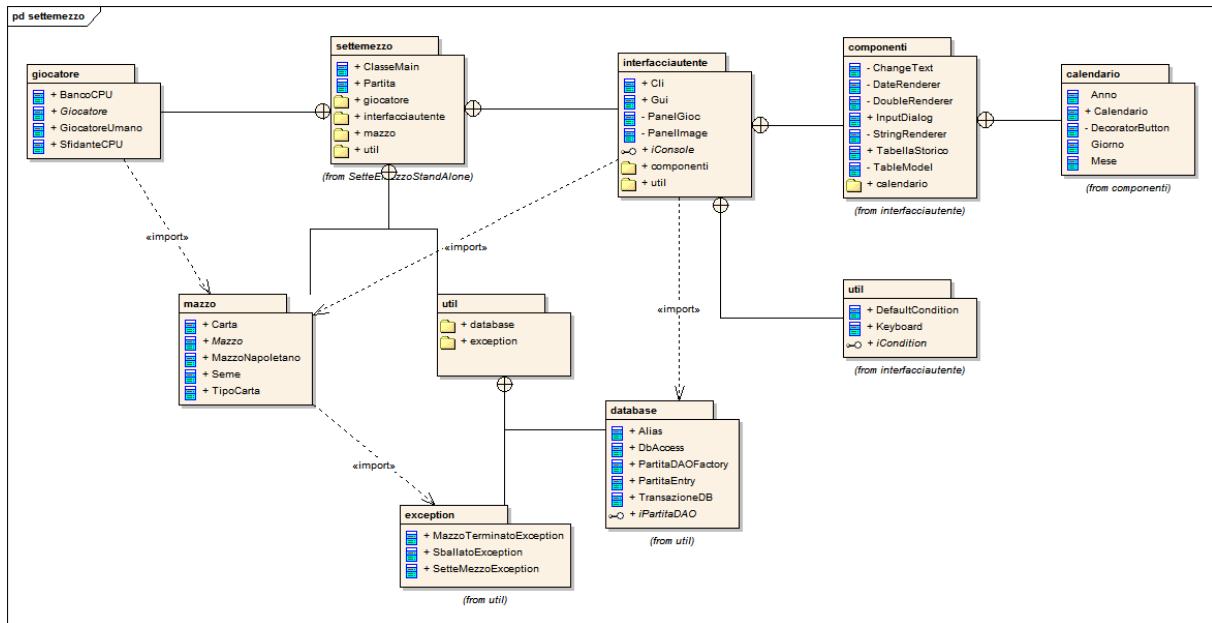
Modifiche sono evidenti nella sezione riguardante l’interazione con il database, attraverso l’implementazione di un’interfaccia [DAO](#) e di una Factory. A questo bisogna aggiungere l’adozione di un file [XML](#) per la memorizzazione dei dati di accesso al DBMS.

Sul fronte dell’interfaccia utente si nota l’eliminazione della classe *IntegerCondition* in quanto non più necessaria con la specifica implementata; evidenziamo l’introduzione di classi per la visualizzazione di un help in linea e di uno storico in formato tabellare. La struttura della classe adibita alla visualizzazione della parte grafica del gioco è stata ristrutturata in modo da ottenere un’applicazione semplice da usare.

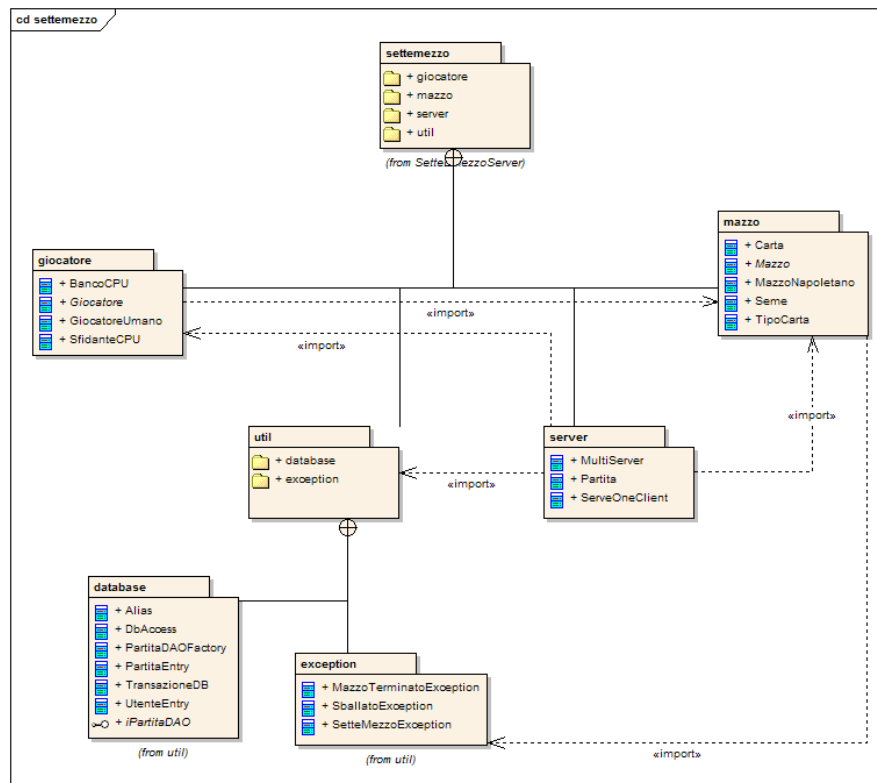
Il progetto si valorizza anche per la presenza di un calendario adottato per la richiesta di una data da utilizzare per la visualizzazione dello storico delle partite giocate. Valore aggiunto è anche l’introduzione della classe *Mediatore* che si occupa della gestione degli eventi scatenati dall’interfaccia grafica e l’utilizzo di classi [JSP](#) per la realizzazione di un sistema di registrazione e di visualizzazione dello storico personale.

Presentiamo adesso i diagrammi di package delle versioni implementate.

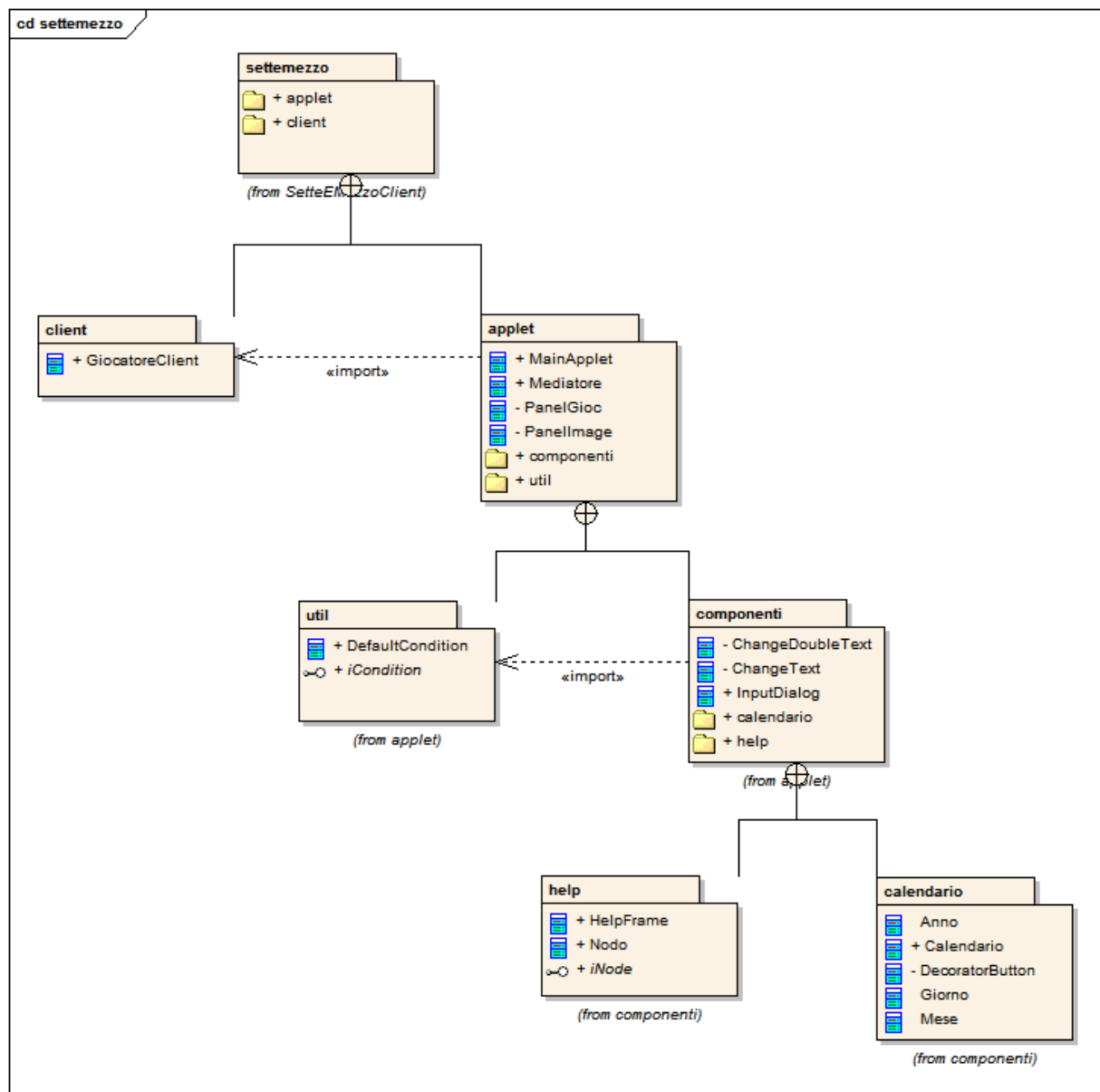
**FIGURA 1- DIAGRAMMA DI PACKAGE DEL PRODOTTO SETTE E MEZZO STAND ALONE**



**FIGURA 2- DIAGRAMMA DI PACKAGE DEL PRODOTTO SETTE E MEZZO CLIENT/SERVER (LATO SERVER)**



**FIGURA 3- DIAGRAMMA DI PACKAGE DEL PRODOTTO SETTE E MEZZO CLIENT/SERVER (LATO CLIENT)**



## 6.1. Modello di decisione

La tabella riassume la suddivisione dei package per prodotto. Inoltre evidenzia i package che sono stati riusati per il prodotto Client/Server.

Package	Prodotti		
	Stand Alone	Client/Server	
		Lato client	Lato server
settemezzo.giocatore	✓		✓
settemezzo.mazzo	✓		✓
settemezzo.util.exception	✓		✓
settemezzo.util.database	✓		✓
settemezzo	✓		
settemezzo.interfacciautente	✓		
settemezzo.interfacciautente.util	✓		
settemezzo.interfacciautente.componenti	✓		
settemezzo.interfacciautente.componenti. calendario*	✓		
settemezzo.server			✓
settemezzo.client		✓	
settemezzo.applet		✓	
settemezzo.applet.util		✓	
settemezzo.applet.componenti		✓	
settemezzo.applet.componenti.help		✓	
settemezzo.applet.componenti.calendario *		✓	

**\*N.B.:** si fa notare che i package “settemezzo.applet.componenti.calendario” e i package “settemezzo.interfacciautente.componenti.calendario”, anche se presentano nomi differenti, in realtà contengono le stesse classi che non sono state modificate. Il diverso nome è giustificato dal fatto che appartengono a prodotti differenti con differenti interfacce grafiche.

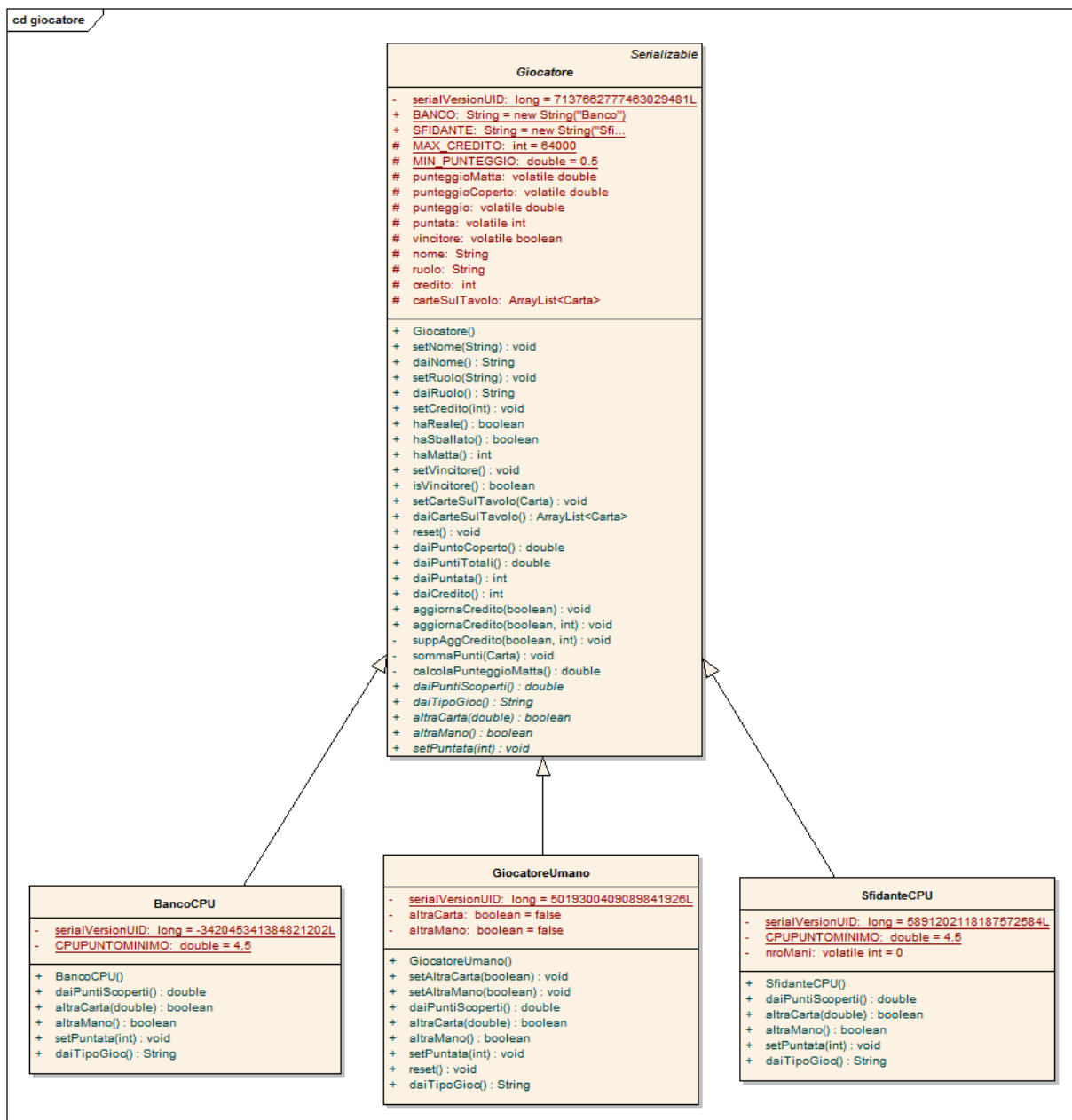


## 6.2. Specifica dei package e delle classi

Analizziamo singolarmente i package implementati, evidenziando le differenze rispetto al diagramma di package base, cioè il progetto proposto dal docente.

### 6.2.1. settemezzo.giocatore

Il package contiene le classi che rappresentano i possibili ruoli previsti per un giocatore del gioco Sette e Mezzo. In esso troviamo le classi che raffigurano il comportamento del giocatore avversario gestito dalla cpu nel ruolo di banco e sfidante (rispettivamente *BancoCPU* e *SfidanteCPU*) e la classe che rappresenta il giocatore della partita (*GiocatoreUmano*). Tutte le classi derivano da una classe base (*Giocatore*) che presenta i tratti comuni dei due tipi di giocatori. Questo package, comune in entrambi i prodotti, non ha subito modifiche rispetto alla struttura proposta.



### 6.2.1.1. Giocatore

*Giocatore* è una classe astratta che consente di definire non solo il comportamento comune di ogni giocatore, ma anche quei comportamenti la cui implementazione risulta essere differente a seconda del tipo e del ruolo del giocatore in esame (metodi astratti). Nella classe sono salvate tutte le informazioni necessarie alla rappresentazione di un giocatore: il nome, il ruolo assunto durante una partita, il credito, le carte ricevute dal mazziere durante una mano, la sua puntata, il punteggio totale e quello della carta coperta. Il credito di un giocatore è impostato al valore massimo di 64000 €.

Differenza evidente, rispetto alla specifica proposta, è l'esclusione del campo video di tipo *iConsole*, consentendo una separazione tra il livello di presentazione e il livello di dominio. Con questa soluzione la classe *Giocatore* memorizza le informazioni sul giocatore, e non interagisce direttamente con l'utente.

Ulteriore differenza è l'introduzione del campo ruolo necessario alla memorizzazione del ruolo del giocatore assunto durante una mano, che può essere banco o sfidante. Questa variabile è d'ausilio nello scambio dei ruoli.

Rispetto alla specifica di base è stato aggiunto il campo *carteSulTavolo*, che consente la memorizzazione di tutte le carte ricevute dal mazziere in una mano, e il campo *punteggioMatta*. Il primo facilita la determinazione del punteggio assunto dalla matta, il quale influenza il punteggio complessivo, e risulta d'aiuto nella determinazione del vincitore di una mano, nel caso entrambi i giocatori abbiano totalizzato il punteggio massimo. La struttura adottata per la memorizzazione delle carte è stata un *ArrayList*, in quanto non si può sapere a priori il numero di carte che richiederà il giocatore.

Il campo *punteggioMatta*, invece, conserva il precedente punteggio della matta e viene usato nell'algoritmo di determinazione del punteggio totale.

Rimangono inalterate, rispetto al progetto del docente, le definizioni dei metodi astratti *altraCarta()* e *altraMano()*, mentre vengono sostituiti i restanti. Gli altri metodi astratti definiti sono:

- *daiPuntiScoperti()*, restituisce il punteggio relativo alle carte scoperte.
- *daiTipoGioc()*, che restituisce il tipo del giocatore; cioè ci informa se il giocatore è un giocatore umano o comandato dalla cpu.
- *setPuntata(int puntata)*, che imposta la puntata del giocatore.

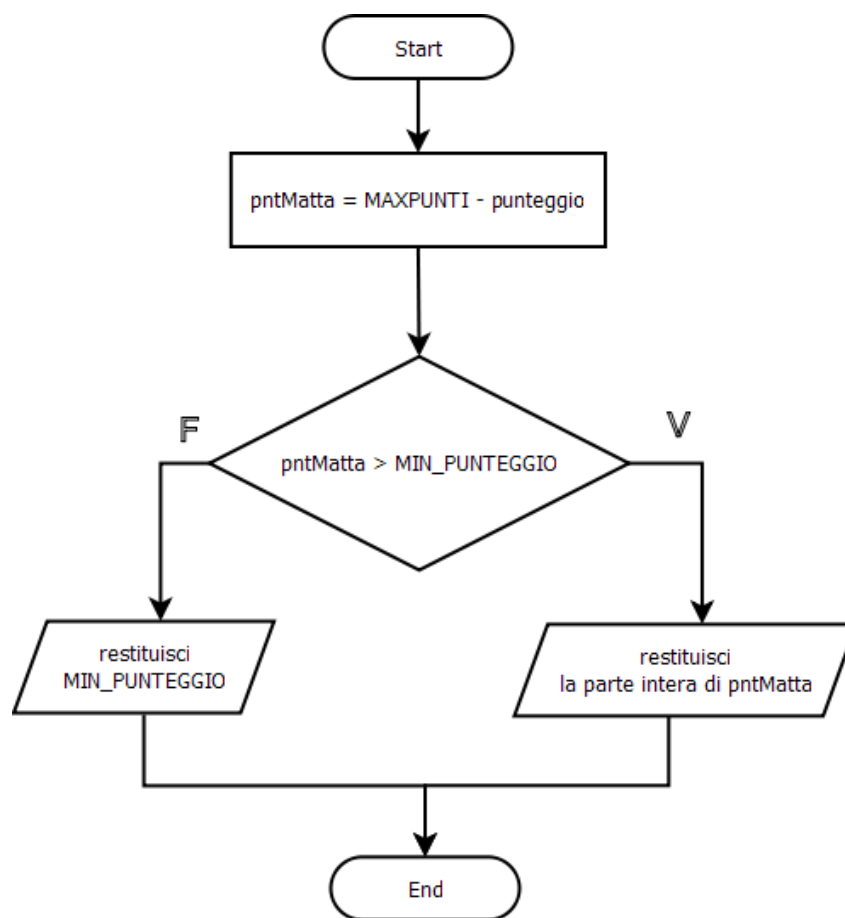
Analizziamo adesso due algoritmi importanti presenti nella classe in esame.

### Algoritmo del calcolo del punteggio della matta

Questo algoritmo, implementato dal metodo privato *calcolaPunteggioMatta()*, consente la determinazione del punteggio della carta che svolge il ruolo di matta.

Viene inizialmente determinato il punteggio che dovrà assumere la matta, sottraendo al massimo punteggio realizzabile il punteggio totalizzato dal giocatore. In seguito si verifica il valore ottenuto. Se risulta essere maggiore al minimo punteggio ottenibile (0.5), la matta varrà il valore intero del risultato calcolato, in quanto la matta può assumere i valori interi compresi tra 1 e 7. Altrimenti verrà restituito il minimo punteggio, cioè la matta viene intesa come una semplice figura.

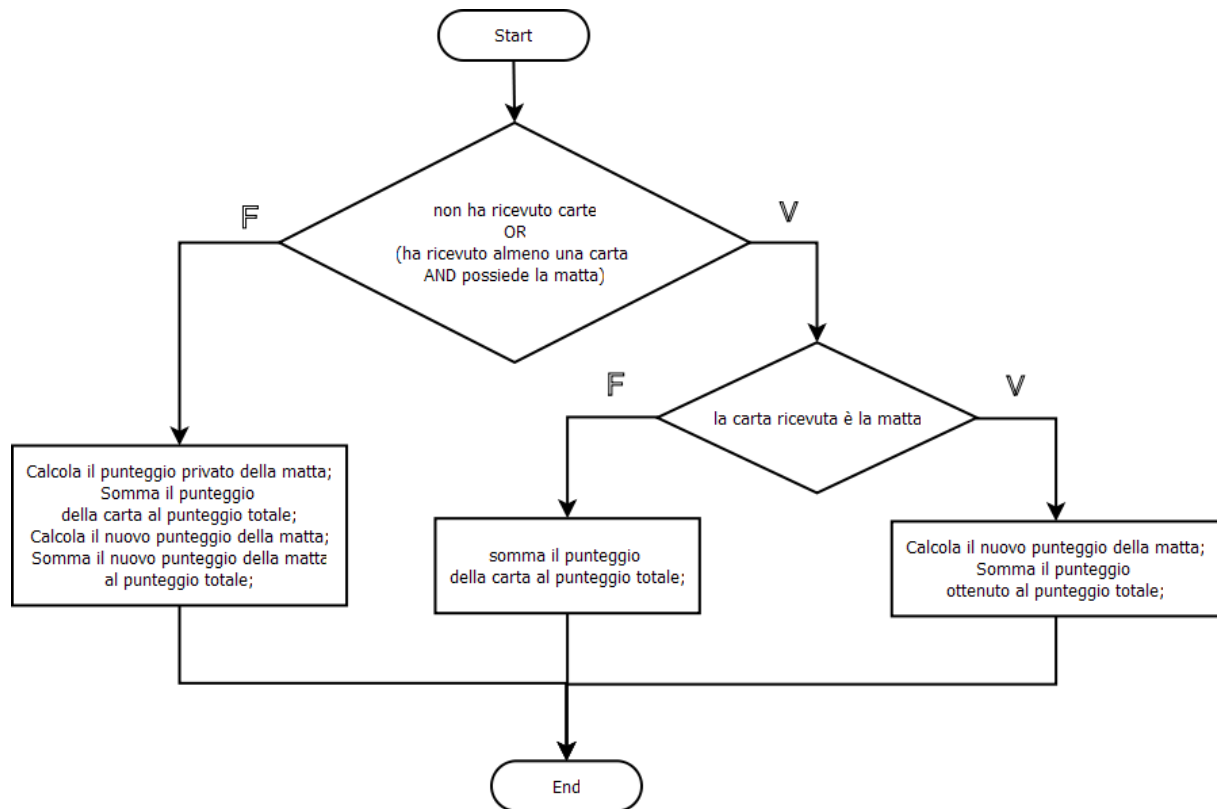
Il diagramma di flusso che descrive l'algoritmo è sotto riportato.



### Algoritmo del calcolo del punteggio totale

Questo algoritmo, implementato dal metodo privato `sommaPunti(Carta carta)`, consente la determinazione del punteggio totale.

Il diagramma di flusso che descrive l'algoritmo è sotto riportato.



#### 6.2.1.2. **GiocatoreUmano**

La classe in esame rappresenta l'utente che utilizza il gioco. In base al gioco potrà impersonare sia il ruolo di banco sia il ruolo di sfidante. Estende la classe *Giocatore* e quindi implementa i relativi metodi astratti.

#### 6.2.1.3. **BancoCPU**

Questa classe implementa la logica del banco quando il computer assume tale ruolo. Estende la classe *Giocatore*.

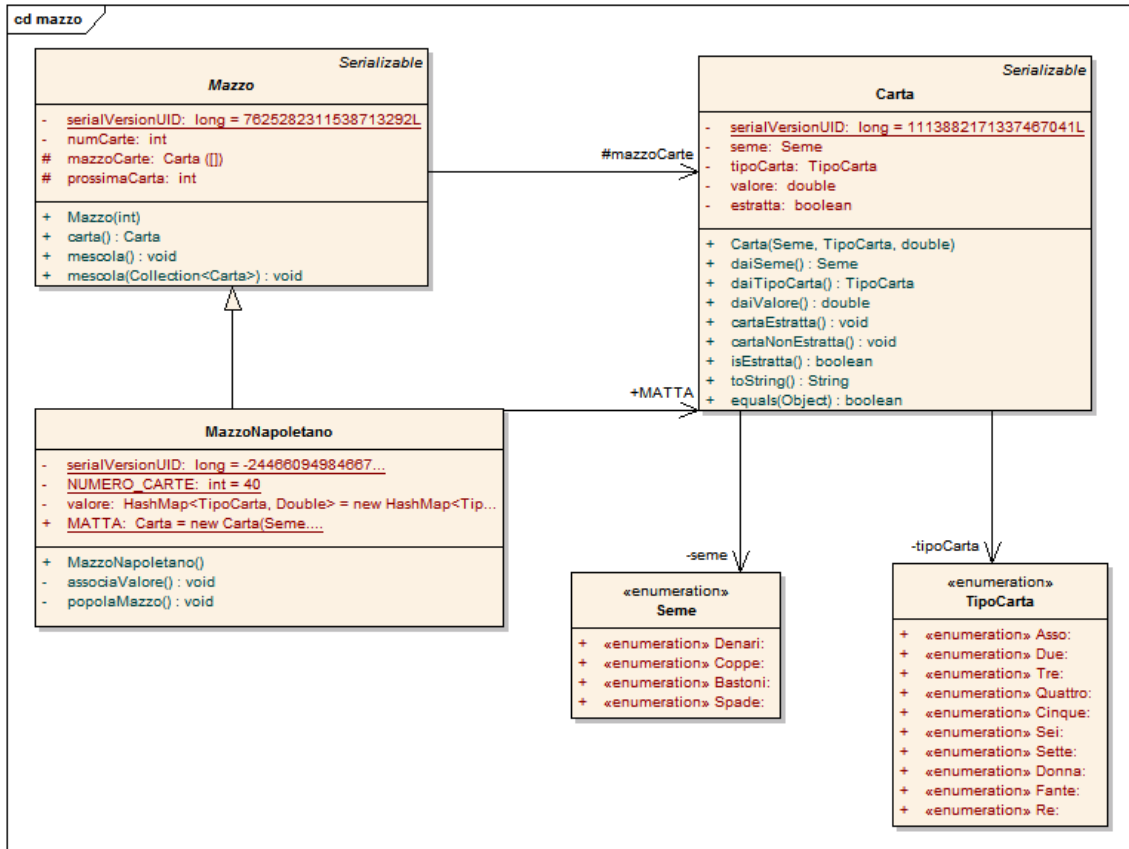
#### 6.2.1.4. **SfidanteCPU**

Questa classe implementa la logica dello sfidante quando impersonato dal computer. Anch'essa estende la classe *Giocatore*. Diversa rispetto alla specifica proposta è l'implementazione del metodo astratto *altraMano()*: il numero delle mani da giocare rimane pari a quattro, ma la variabile *nroMani* adibita a tener traccia delle mani presenta come valore iniziale zero. Viene, quindi, prima incrementato *nroMani* e restituito vero se inferiore a cinque, altrimenti viene restituito falso.

### 6.2.2. settemezzo.mazzo

Presenta al suo interno le classi necessarie alla definizione di un mazzo di carte. È costituito da una classe astratta *Mazzo* che offre metodi basilari per la definizione di un qualsiasi mazzo di carte, una classe che estende quest'ultima, che specifica il tipo di mazzo necessario per il gioco del Sette e Mezzo (*MazzoNapoletano*), una classe le cui istanze rappresentano ogni singola carta del mazzo (*Carta*), e due classi enumerative di ausilio nella definizione delle carte (*Seme* e *TipoCarta*).

Rispetto alla struttura suggerita, l'unico cambiamento effettuato è stata l'introduzione di una classe base da cui partire per la creazione di un mazzo di carte da gioco.



### 6.2.2.1. Mazzo

La classe realizza un mazzo di carte generiche opportunamente manipolabile.

Rispetto alla specifica definita, la classe *Mazzo* è stata modificata in una classe astratta, in modo tale da consentire la definizione di una struttura comune utilizzabile nella creazione di un qualunque mazzo di carte.

Nella classe sono memorizzate le informazioni necessarie alla rappresentazione e alla gestione di un mazzo di carte: il numero di carte nel mazzo, le carte che lo compongono e una variabile usata per tener traccia della prossima carta distribuibile. Per la memorizzazione delle carte la struttura adottata è stata quella di un vettore al posto di un *ArrayList* come proposto dalla specifica. Sia la struttura *ArrayList* che il vettore permettono di recuperare un elemento in una determinata posizione e di aggiungere o rimuovere in coda al costo di  $O(1)$ . Aggiungere o rimuovere elementi in una qualsiasi altra posizione ha costo lineare  $O(n)$ , in quanto è necessario lo shift (rispettivamente in avanti o in indietro) degli elementi successivi.

Visto che l'operazione dominante risulta l'aggiunta in coda, effettuata solo in fase di creazione del mazzo, e che la dimensione del mazzo risulta costante durante tutto il suo utilizzo, si è ritenuto più appropriato usare il vettore rispetto ad una struttura a dimensione variabile.

I metodi che compongono la classe sono:

- *Mazzo(int dimensione)*, consente la istanziazione di un mazzo di carte con dimensione passata come parametro.
- *Carta carta()*, restituisce la prossima carta estratta dal mazzo. Questo metodo riassume il compito svolto dai metodi *carta()* e *daiCarta()* presenti nella specifica base: si è ritenuto più semplice avere un solo metodo che fornisca la carta estratta e aggiorni il riferimento alla prossima carta che avere due metodo da eseguire in sequenza.
- *mescola()*, consente di mescolare il mazzo di carte, resettando il loro stato.
- *mescola(Collection<Carta> carteSulTavolo)*, consente di mescolare il mazzo di carte, lasciando inalterato lo stato delle carte passate come parametro. Consente quindi un mescolamento parziale, mescolando il mazzo delle carte estratte ma non in uso nella mano.

Analizziamo adesso i tre algoritmi importanti presenti nella classe in esame.

### Algoritmo della distribuzione della prossima carta

Il metodo `carta()` che implementa tale algoritmo, si presenta realizzato tramite un metodo ricorsivo. Si verifica se l'indice della prossima carta da estrarre è maggiore del numero di carte: in caso di esito positivo si solleva l'eccezione `MazzoTerminatoException`; altrimenti se la prossima carta è stata estratta, quindi risulta già utilizzata, si incrementa il riferimento alla prossima carta e si riesegue il metodo.

Altrimenti, vuol dire che la carta non risulta estratta. Dunque si segna la carta come estratta, si incrementa l'indice della prossima carta e ritorna la carta appena estratta.

Presentiamo anche l'implementazione iterativa di tale metodo.

```
public Carta carta() throws MazzoTerminatoException {
    boolean trovata = false;
    Carta cartaTrovata = null;

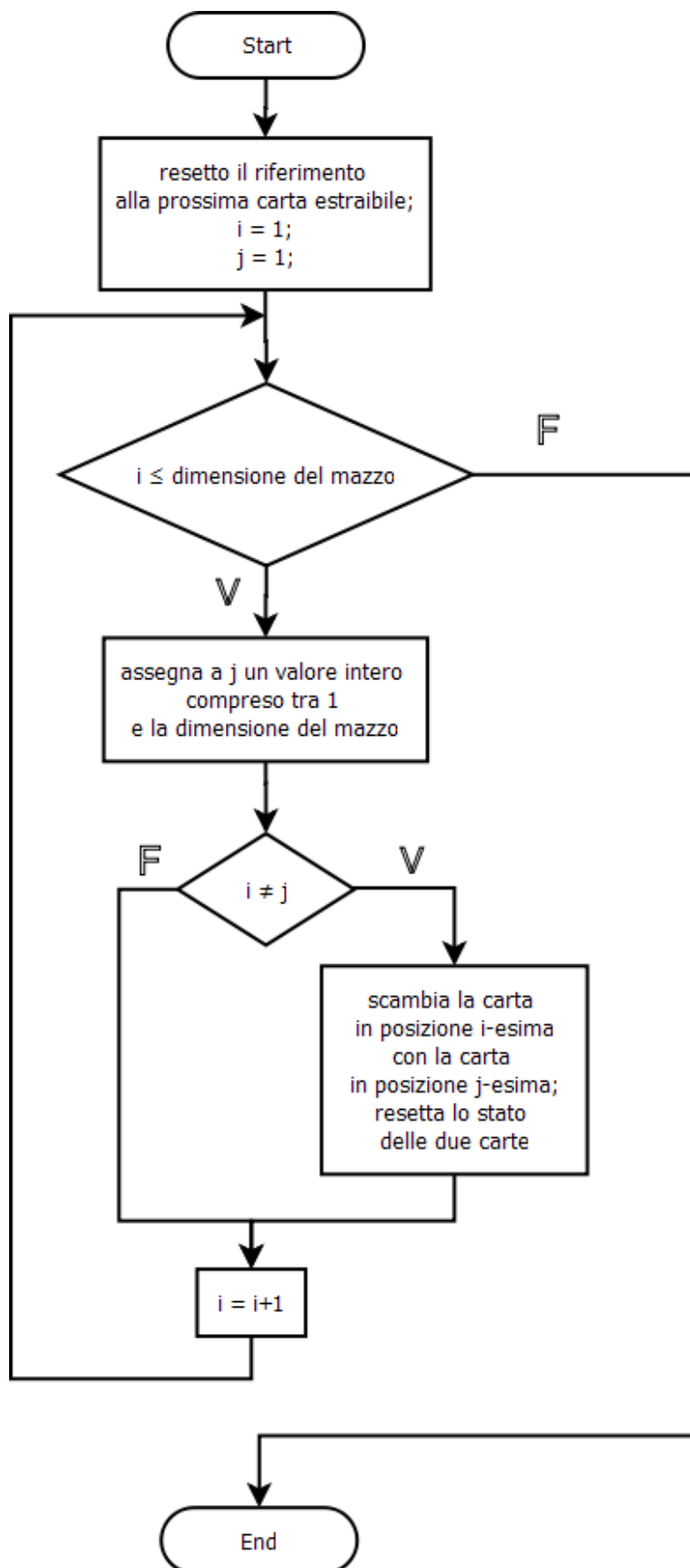
    while(!trovata)
    {
        if(prossimaCarta >= numCarte)
            throw new MazzoTerminatoException();
        else if(mazzoCarte[prossimaCarta].isEstratta())
            prossimaCarta++;
        else
        {
            trovata = true;
            cartaTrovata = mazzoCarte[prossimaCarta];
            cartaTrovata.cartaEstratta();
        }
    }

    return cartaTrovata;
}
```



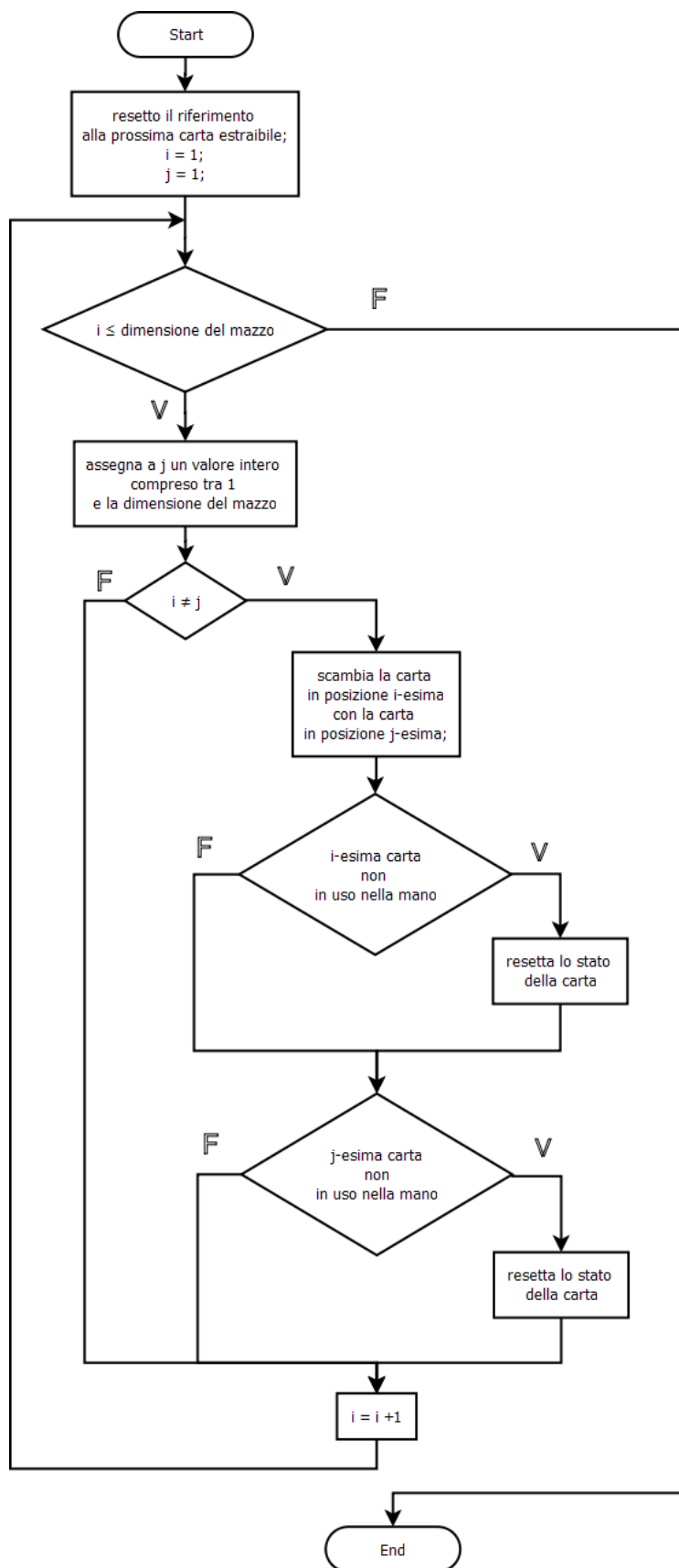
### Algoritmo di mescolamento delle carte

Ecco l'algoritmo implementato dal metodo *mescola()*.



## Algoritmo di mescolamento parziale delle carte

Ecco l'algoritmo implementato dal metodo *mescola(Collection carteSulTavolo)*.



#### 6.2.2.2. **MazzoNapoletano**

Rappresenta un mazzo di carte napoletane opportunamente manipolabile per il gioco del Sette e Mezzo. La classe estende la classe astratta *Mazzo*.

*MazzoNapoletano* presenta al suo interno un campo pubblico che definisce la carta che assume il ruolo di mazzo. Non definisce ulteriori metodi pubblici, ma definisce due metodi privati usati nella creazione del mazzo di carte:

- *associaValore()*, che consente di associare ad ogni tipo di carta il relativo valore.
- *popolaMazzo()*, consente la creazione del mazzo napoletano di quaranta carte attraverso l'avvaloramento di ciascuna carta da gioco componente.

#### 6.2.2.3. **Carta**

La classe riproduce una singola carta da gioco dotata di seme, tipo e valore. Inoltre è stato aggiunto un'ulteriore campo booleano che consente di tener traccia delle carte estratte dal mazzo.

Il cambio di stato della carta avviene grazie ai seguenti metodi:

- *cartaEstratta()*, che permette di impostare la carta come estratta dal mazzo.
- *cartaNonEstratta()*, che permette di impostare lo stato della carta come non estratta. Utile in fase di mescolamento del mazzo.
- *isEstratta()*, che consente invece di verificare lo stato della carta.

Si è ritenuto necessario effettuare overriding del metodo *equals()* presente nella classe *Object*; tale metodo viene indirettamente utilizzato nel programma in quanto si fa uso dei metodi *indexOf()* della classe *ArrayList* e del metodo *contains()* definito nell'interfaccia *Collection*. Il primo metodo viene utilizzato nella classe *Giocatore* per l'individuazione della mazzo, il secondo nella classe *Mazzo* per un mescolamento parziale del mazzo da gioco.

I metodi riguardanti il recupero delle informazioni di una carta non hanno subito alcuna modifica.

#### 6.2.2.4. **Seme**

La classe *Seme* è una classe di ausilio nella definizione di una carta napoletana: presenta al suo interno i quattro semi presenti in un mazzo di carte napoletano. Non ha subito modifiche rispetto alla specifica stabilita.

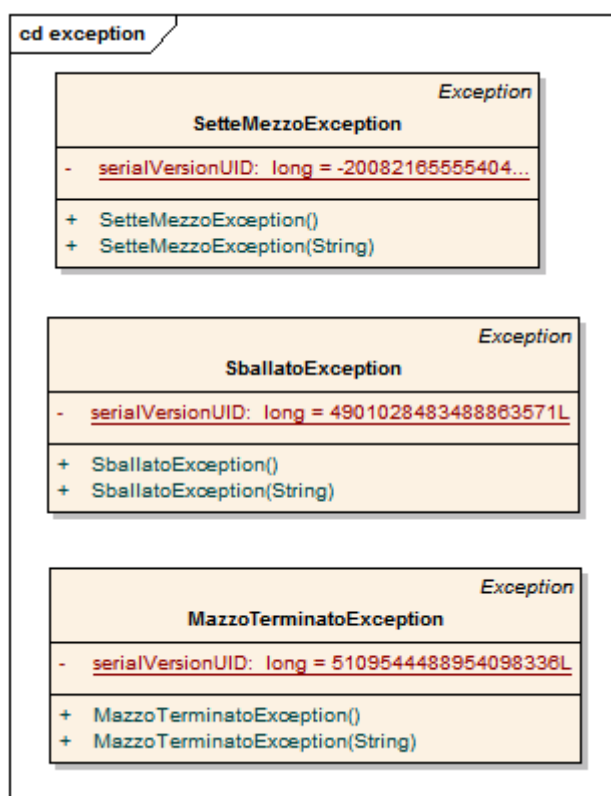
#### 6.2.2.5. **TipoCarta**

La classe *TipoCarta* è anch'egli di ausilio nella definizione di una carta napoletana: presenta al suo interno i tipi presenti in un mazzo di carte napoletano. Non ha subito modifiche rispetto alla specifica stabilita.

### 6.2.3. `settemezzo.util.exception`

Il package presenta al suo interno le eccezioni personalizzate necessarie nei prodotti. Le eccezioni definite sono state realizzate per gestire gli eventi “fine delle carte” (*MazzoTerminatoException*), “punteggio totalizzato pari a sette e mezzo” (*SetteMezzoException*) e “punteggio totalizzato superiore a sette e mezzo” (*SballatoException*).

Rispetto alla soluzione base cambia il nome del package, che si sarebbe dovuto chiamare “*settemezzo.util*”; inoltre la classe *Keyboard* è stata spostata in un altro package e sono stati introdotti gli eventi riguardanti il punteggio.



#### 6.2.3.1. *MazzoTerminatoException*

La classe crea un'eccezione usata per gestire il caso in cui il mazzo di carte sia terminato. L'eccezione personalizzata estende la classe *Exception* e ne sovrascrive il costruttore parametrizzato e quello di default.

#### 6.2.3.2. *SballatoException*

La classe crea un'eccezione usata per gestire il caso in cui un giocatore abbia totalizzato un punteggio superiore al punteggio massimo. Anch'essa sovrascrive il costruttore parametrizzato e quello di default.

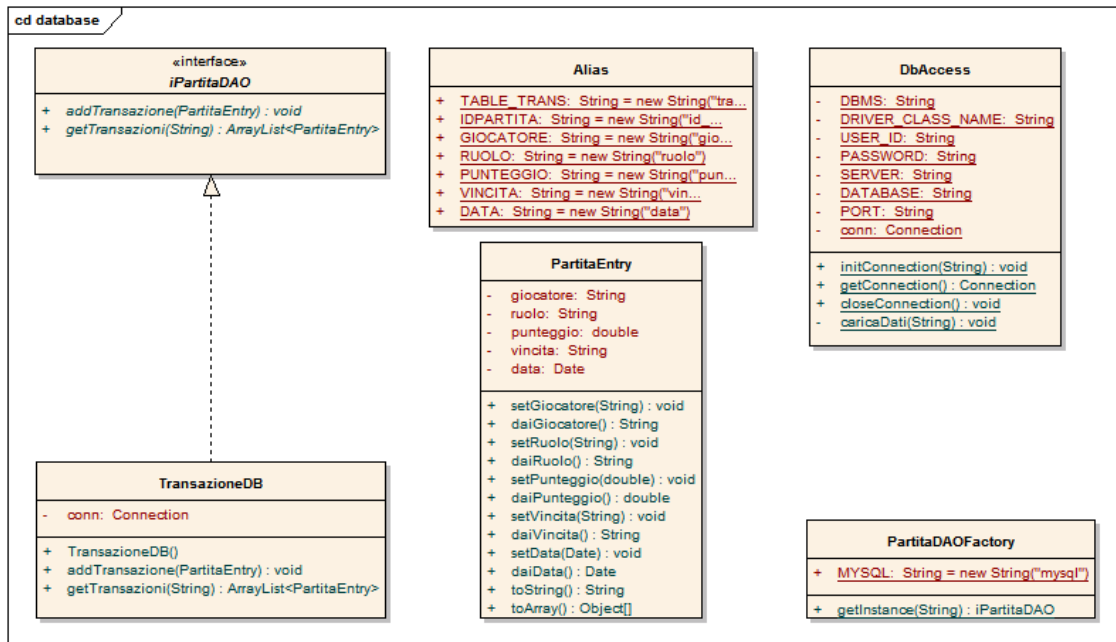
#### 6.2.3.3. *SetteMezzoException*

La classe crea un'eccezione usata per gestire il caso in cui un giocatore abbia totalizzato un punteggio pari al punteggio massimo.

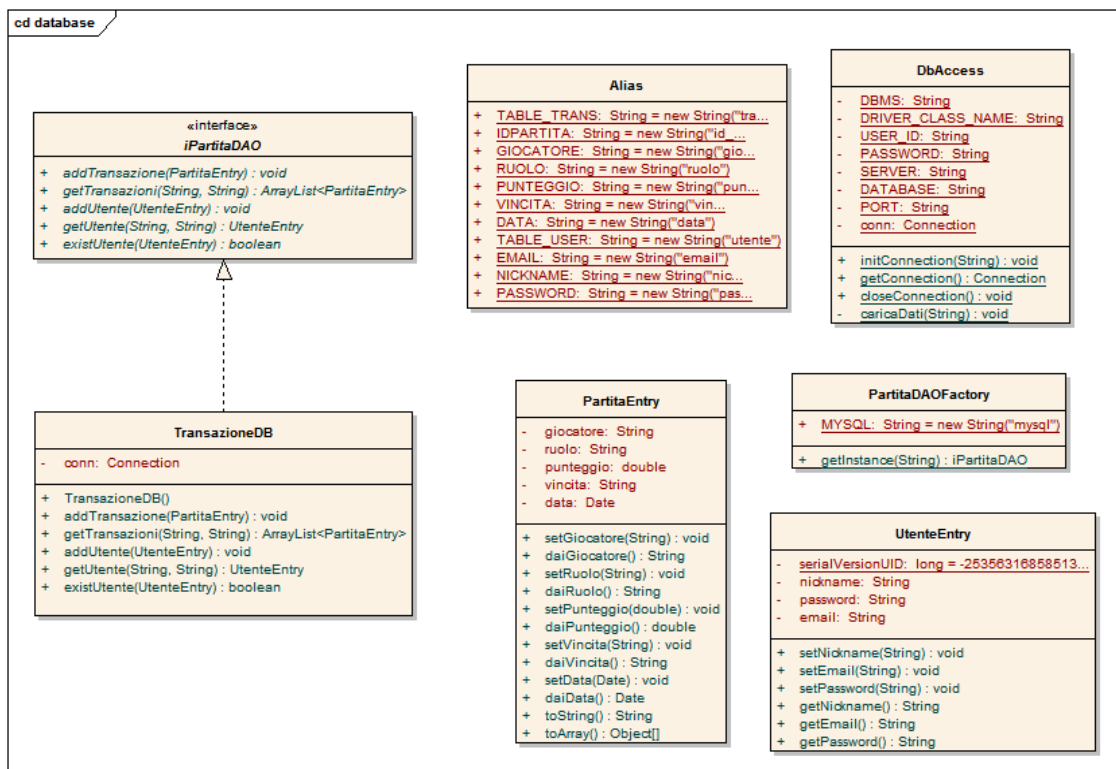
#### 6.2.4. settemezzo.util.database

Consente l'interfacciamento con una base di dati per mantenere traccia della partita giocata. Secondo quando definito nel progetto base, il package doveva permettere l'interfacciamento solo con il DBMS MySQL, ma con l'introduzione di ulteriori classi, il package consente l'interfacciamento con qualsiasi DBMS.

Rispetto al progetto base sono state introdotte quattro nuove classi: *iPartitaDAO*, *PartitaDAOFactory*, *PartitaEntry* ed *Alias*.



Nella versione Client/Server, il package presenta un'ulteriore classe per la modellazione di un utente registrato (*UtenteEntry*).



#### 6.2.4.1. Alias

La classe *Alias* presenta al suo interno tutti i nomi delle tabelle, e relativi attributi, che costituiscono il database. È utilizzata come classe di ausilio nella composizione delle query. Grazie a questa classe, i cambiamenti apportati al database (quali il cambio dei nomi degli attributi o delle tabelle) non si ripercuotono sulle query, che rimangono inalterate. Tale soluzione estende e migliora la specifica base.

Nella versione presente nel prodotto Stand Alone, la classe presenta il nome della tabella “*transazione*” adibita al salvataggio di una mano, e relativi attributi:

- *id\_partita*, identificativo univoco della partita giocata, nonché chiave primaria della tabella.
- *giocatore*, il nome del giocatore che ha partecipato alla partita.
- *ruolo*, il ruolo assunto dal giocatore in esame (banco o sfidante).
- *punteggio*, il punteggio totalizzato dal giocatore in esame.
- *vincita*, la vincita del giocatore in esame. La vincita se preceduta da un segno “-” indica che il giocatore ha perso tale valore, se preceduta da un segno “+” indica che ha vinto tale valore.
- *data*, la data in cui si svolge la partita.

Nella versione presente nel prodotto Client/Server, la classe presenta la tabella “*transazione*” e la tabella “*utente*” adibita al salvataggio delle credenziali degli utenti registrati:

- *email*, l’email dell’utente registrato e anche chiave della tabella.
- *nickname*, il nome utente scelto dall’utente, usato nella visualizzazione del nome del giocatore.
- *password*, la password scelta dall’utente.

#### 6.2.4.2. DbAccess

Consente la connessione ad una base di dati, ricavando il dati per la connessione da un file [XML](#).

La specifica iniziale prevedeva la memorizzazione dei dati necessari alla connessione all'interno della classe. Questa soluzione, sicuramente, comportava la ricompilazione della classe ad ogni cambiamento dei dati di accesso.

Al contrario, l'adozione di un file [XML](#) consente al programmatore il massimo riutilizzo di tale classe consentendo di accedere facilmente a qualunque database. Inoltre il cambiamento dei dati nel file [XML](#) non comporterà alcuna modifica nel codice sorgente della classe che, perciò, non necessita di essere ricompilato.

Il file [XML](#) utilizzato rispetta la seguente [DTD](#), definita dalla Sun stessa.

```
<!-- Copyright 2006 Sun Microsystems, Inc. All rights reserved. -->
<!-- DTD for properties -->
<!ELEMENT properties ( comment?, entry* ) >
<!ATTLIST properties version CDATA #FIXED "1.0">
<!ELEMENT comment (#PCDATA) >
<!ELEMENT entry (#PCDATA) >
<!ATTLIST entry key CDATA #REQUIRED>
```

Un file [XML](#) così strutturato potrà essere facilmente creato e letto da un programma Java.

I metodi implementati nella classe si presentano essere gli stessi definiti nella specifica base, tranne per il metodo *initConnection()* che presenta un parametro di input: il nome del file [XML](#) in cui sono salvate le informazioni.

#### 6.2.4.3. iPartitaDAO

Interfaccia che definisce i metodi d'interazione al database che conserva i salvataggi della partita e gli utenti registrati al sistema. Non presente nella specifica base, è stata aggiunta per aumentare la riusabilità e la manutenibilità del codice.

La classe svolge il ruolo di interfaccia di accesso ai dati (DAO), definendo i metodi utilizzabili per interrogare il database. Nel contempo, essa permette alle classi che la implementano l'utilizzo di una logica completamente propria per la selezione dei dati. Questo consente due grandi vantaggi:

- Nell'applicazione di lavorare soltanto con l'interfaccia: l'implementazione concreta rimane nascosta. Ciò comporta una manutenzione delle applicazioni più facile, poiché la classe che implementa l'accesso ai dati può essere modificata senza che il codice applicativo debba essere rivisto.
- Gli accessi al database sono mediati da una classe centrale: se il modello di dati nel database cambia, l'adattamento riguarderà soltanto questa classe centrale.

L'interfaccia, anche se comune ad entrambi i prodotti, presenta dei metodi non comuni che andremo ad individuare con la tabella sottostante.

Metodi	Prodotti		
	Stand Alone	Client / Server	
		Lato client	Lato server
addTransazione()	✓		✓
getTransazioni()	✓		✓
addUtente()			✓
getUtente()			✓
existUtente			✓

L'interfaccia *iPartitaDAO* definisce, nella versione Stand Alone, due metodi. Il metodo *addTransazione()* serve per aggiungere un nuovo record nel database. Il metodo *getTransazioni()* recupera lo storico delle partite in una determinata data.

Nella versione Client/Server sono presenti ulteriori tre metodi. Il metodo *addUtente()* consente la memorizzazione di un nuovo utente nel database. Il metodo *getUtente()* permette il recupero delle informazioni di un utente registrato. Il metodo *existUtente()* verifica l'esistenza di un utente nel database.



#### 6.2.4.4. PartitaDAOFactory

La classe consente la creazione di istanze di *iPartitaDAO* a seconda del dbms utilizzato; per far questo implementa il design pattern Factory.

La Factory consente l'istanziamento di un'istanza di accesso ai dati senza tener conto dell'implementazione concreta che verrà utilizzata. La classe risulta, nel nostro caso, molto semplice poiché tiene conto di una sola fonte di dati da collegare, ma con delle semplici modifiche è possibile ampliare il numero di database con cui interagire.

La classe si presenta costituita da un solo metodo, il metodo *getInstance()* che consente di ottenere una classe di comunicazione con il database passato come parametro. I nomi dei dbms con cui è possibile collegarsi sono presenti nella classe stessa sotto forma di campi statici. Anche questa classe rientra in un progetto di miglioramento della specifica base.

#### 6.2.4.5. PartitaEntry

La classe *PartitaEntry* non dispone di una propria logica elaborativa, ma si limita a contenere un record della tabella “transazione” ed a permettere un accesso strutturato ad esso. Inoltre, definisce metodi di supporto per la conversione del record in una stringa o in un vettore di stringhe. La sua creazione si è ritenuta necessaria in seguito alla scelta di scindere i ruoli assegnati alla classe *TransazioneDB*.

#### 6.2.4.6. TransazioneDB

*TransazioneDB* si occupa dell'implementazione dell'interfaccia [DAO](#) e quindi dell'effettiva interazione con il database.

Rispetto alla specifica base, la classe avrebbe dovuto occuparsi di una interazione completa con il database, dalla connessione all'esecuzione della query desiderata. E inoltre doveva anche svolgere il ruolo di rappresentazione dei record delle tabelle del database. Per aumentare il riuso e facilitare la manutenzione si è preferito far in modo che questa classe si occupasse solo dell'esecuzione delle query, mentre gli altri ruoli previsti sono stati delegati ad altre classi.

Per poter inserire e richiamare i record del database, ci si è serviti dell'interfaccia *PreparedStatement*, per la creazione di dichiarazioni SQL con parametri.

La scelta di tale interfaccia è motivata dalle seguenti ragioni:

- *sicurezza*: i *PreparedStatement* offrono maggior sicurezza.
- *prestazioni*: di solito i *PreparedStatement* vengono eseguiti in modo più rapido perché le query sono precompilate in funzione del tipo di database e i parametri inviati separatamente.
- *leggibilità*: le classi che usano tale interfaccia sono più leggibili, poiché l'assegnazione dei valori avviene sotto forma di diverse chiamate di metodo.
- *approccio typesafe*: dal momento che i dati sono assegnati mediante metodi Java, è evidente che possono essere utilizzati soltanto i tipi di dato accettati dai metodi.

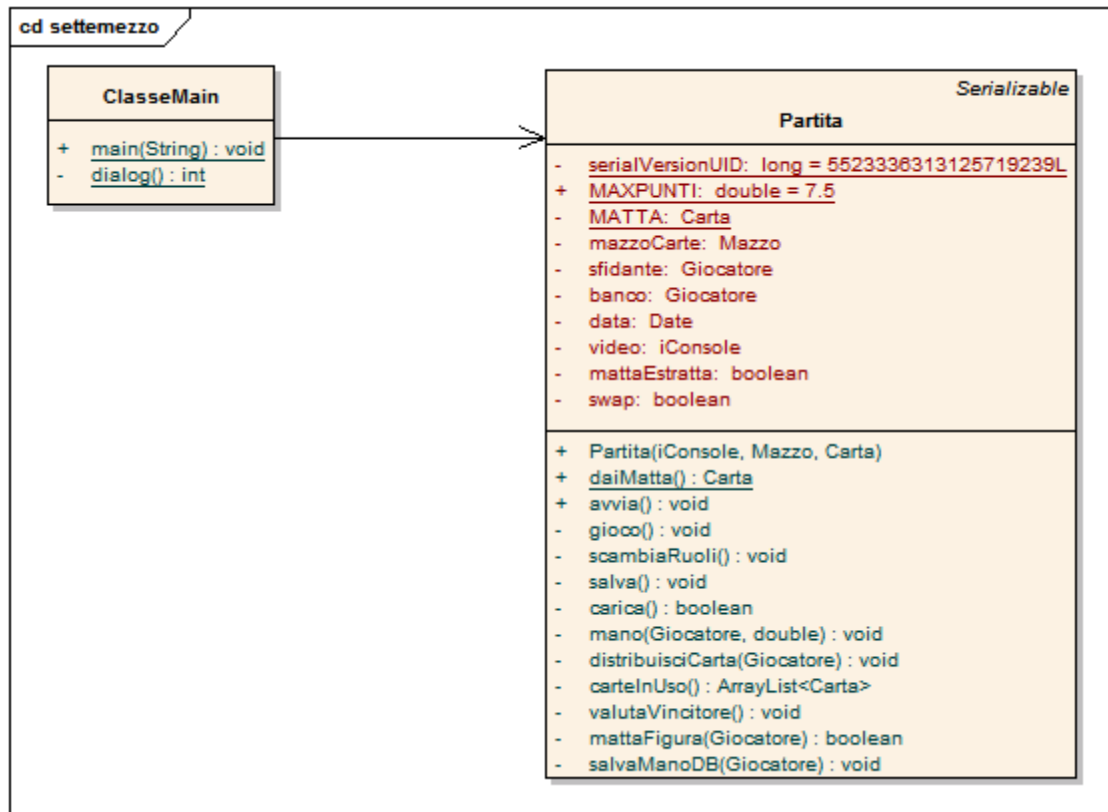
Siccome la classe *iPartitaDAO* cambia con i prodotti, anche la classe *TransazioneDB* cambia con i prodotti. La suddivisione dei metodi rispecchia quella vista nella specifica della classe *iPartitaDAO*.

#### 6.2.4.7. UtenteEntry

Presente solo nella versione Client/Server, la classe si limita a contenere un record della tabella “utente” ed a permettere un accesso strutturato ad esso. Come la classe *PartitaEntry*, non dispone di una logica elaborativa. Anch'essa estende la specifica base.

### 6.2.5. settemezzo

Presente solo nella versione Stand Alone, il package contiene la classe necessaria all'avvio del programma (*ClasseMain*) e quella che gestisce un'intera partita tra due giocatori (*Partita*). Non presenta differenza rispetto alla versione proposta.



#### 6.2.5.1. ClasseMain

*ClasseMain* si occupa dell'avvio del gioco nella versione Stand Alone; si occupa inoltre della visualizzazione di un set di opzioni che consente all'utente di scegliere la modalità con cui giocare la partita.

La classe consente l'avvio di una sola istanza della partita: questo è stato ottenuto con l'utilizzo di una socket. Viene creata una *ServerSocket* e gli viene assegnata una porta: ciò assicura che si possa avviare una sola partita su ciascuna macchina. Infatti, se la porta è già in uso dal sistema, non potrà essere creata una nuova socket su quella porta, dunque verrà sollevata l'eccezione relativa a tale problema, e notificato all'utente la presenza di una partita precedentemente avviata.

#### 6.2.5.2. Partita

*Partita* si presenta come la classe di maggior importanza del progetto: infatti in essa si concentra la logica per la gestione di una partita a Sette e Mezzo tra due giocatori. La classe *Partita* è, dunque, comune ad entrambi i prodotti ma differiscono nell'implementazione: la versione Client/Server presenta un'evoluzione della codifica rispetto alla versione Stand Alone. Questa evoluzione è dovuta proprio ad un differente approccio di sviluppo del prodotto Client/Server, più orientato a soddisfare le necessità dell'utente al fine di migliorare l'interazione di quest'ultimo con l'applicazione.

La logica di gioco implementata è quella tipica del gioco del Sette e Mezzo: il giocatore interpreta inizialmente il ruolo di sfidante; si distribuiscono le carte coperte e in seguito si consente di giocare prima allo sfidante e poi al banco. Successivamente si valutano i punteggi conseguiti e si dichiara il giocatore vincitore della mano. A seconda della volontà del giocatore e del suo credito, il gioco continua con un'ulteriore mano oppure termina.

Oltre a contenere la logica di una partita, la classe si occupa della gestione del salvataggio e del caricamento di una partita. Per consentire tali operazioni, si è implementato il meccanismo di serializzazione e di deserializzazione degli oggetti: grazie a questa tecnologia fornita dal linguaggio Java, si riesce a consentire il salvataggio dello stato di una partita e l'eventuale ripresa di essa.

Il metodo *salva()* si occupa della serializzazione dello stato della partita. All'atto del salvataggio, si è deciso di serializzare i soli campi privati della classe necessari alla ripresa della partita: per questa ragione si è deciso di salvare i campi relativi ai due giocatori, il mazzo di carte usato, la relativa matta del mazzo e la variabile *mattaEstratta* che indica se la carta indicata come matta è stata estratta durante la mano. È bene precisare che non tutte le informazioni riguardanti i giocatori vengono salvate: infatti il punteggio e la puntata non vengono salvate, in quanto al caricamento si inizierà una nuova mano della partita salvata e tali valori non verranno utilizzati. Questo è consentito dal modificatore "*transient*".

L'intero stato della partita vengono salvate in un file binario con estensione ".dat".

Il metodo adibito al caricamento della partita precedentemente salvata è *carica()*; esso svolge il ruolo opposto al metodo *salva()*, ripristinando lo stato della partita salvata in modo da consentire la ripresa della partita. Il metodo *salvaManoDB()*, infine, salva nel database le informazioni relative alle mani concluse di qualsiasi giocatore indipendentemente dal ruolo.

### Algoritmo di determinazione del vincitore della mano corrente

Presentiamo l'algoritmo implementato dal metodo `valutaVincitore()`; vista la complessità di tale algoritmo, mostriamo, invece del classico diagramma di flusso, una tabella riassuntiva utilizzata per l'individuazione dei vari casi che si possono presentare.

Caso #	Punteggio banco	Punteggio sfidante	Vincitore	Tipo posta
1	7 ½ Matta e Figura	> 7 ½	Banco	Sfidante paga posta semplice
		< 7 ½	Banco	Sfidante paga posta doppia
		7 ½ con più di 2 carte	Banco	Sfidante paga posta doppia
		7 ½ Sette e Figura	Banco	Sfidante paga posta semplice
2	7 ½ Sette e Matta	> 7 ½	Banco	Sfidante paga posta semplice
		< 7 ½	Banco	Sfidante paga posta doppia
		7 ½ con più di 2 carte	Banco	Sfidante paga posta doppia
		7 ½ Sette e Figura	Banco	Sfidante paga posta semplice
3	7 ½ Sette e Figura	> 7 ½	Banco	Sfidante paga posta semplice
		< 7 ½	Banco	Sfidante paga posta doppia
		7 ½ con più di 2 carte	Banco	Sfidante paga posta doppia
		7 ½ Sette e Figura	Banco	Sfidante paga posta semplice
		7 ½ Matta e Figura	Banco	Sfidante paga posta semplice
		7 ½ Sette e Matta	Banco	Sfidante paga posta semplice
4	7 ½ con più di 2 carte	> 7 ½	Banco	Sfidante paga posta semplice
		< 7 ½	Banco	Sfidante paga posta semplice
		7 ½ con più di 2 carte	Banco	Sfidante paga posta semplice
		7 ½ Sette e Figura	Banco	Sfidante paga posta semplice
		7 ½ Matta e Figura	Sfidante	Sfidante riceve posta doppia
		7 ½ Sette e Matta	Banco	Sfidante paga posta semplice

5	< 7 ½	> 7 ½	Banco	Sfidante paga posta semplice
		< 7 ½ puntBanc ≥ puntSfid	Banco	Sfidante paga posta semplice
		< 7 ½ puntSfid > puntBanc	Sfidante	Sfidante riceve posta semplice
		7 ½ con più di 2 carte	Sfidante	Sfidante riceve posta semplice
		7 ½ Sette e Figura	Sfidante	Sfidante riceve posta doppia
		7 ½ Matta e Figura	Sfidante	Sfidante riceve posta doppia
		7 ½ Sette e Matta	Sfidante	Sfidante riceve posta doppia
6	> 7 ½	< 7 ½	Sfidante	Sfidante riceve posta semplice
		7 ½ con più di 2 carte	Sfidante	Sfidante riceve posta semplice
		7 ½ Sette e Figura	Sfidante	Sfidante riceve posta doppia
		7 ½ Matta e Figura	Sfidante	Sfidante riceve posta doppia
		7 ½ Sette e Matta	Sfidante	Sfidante riceve posta doppia

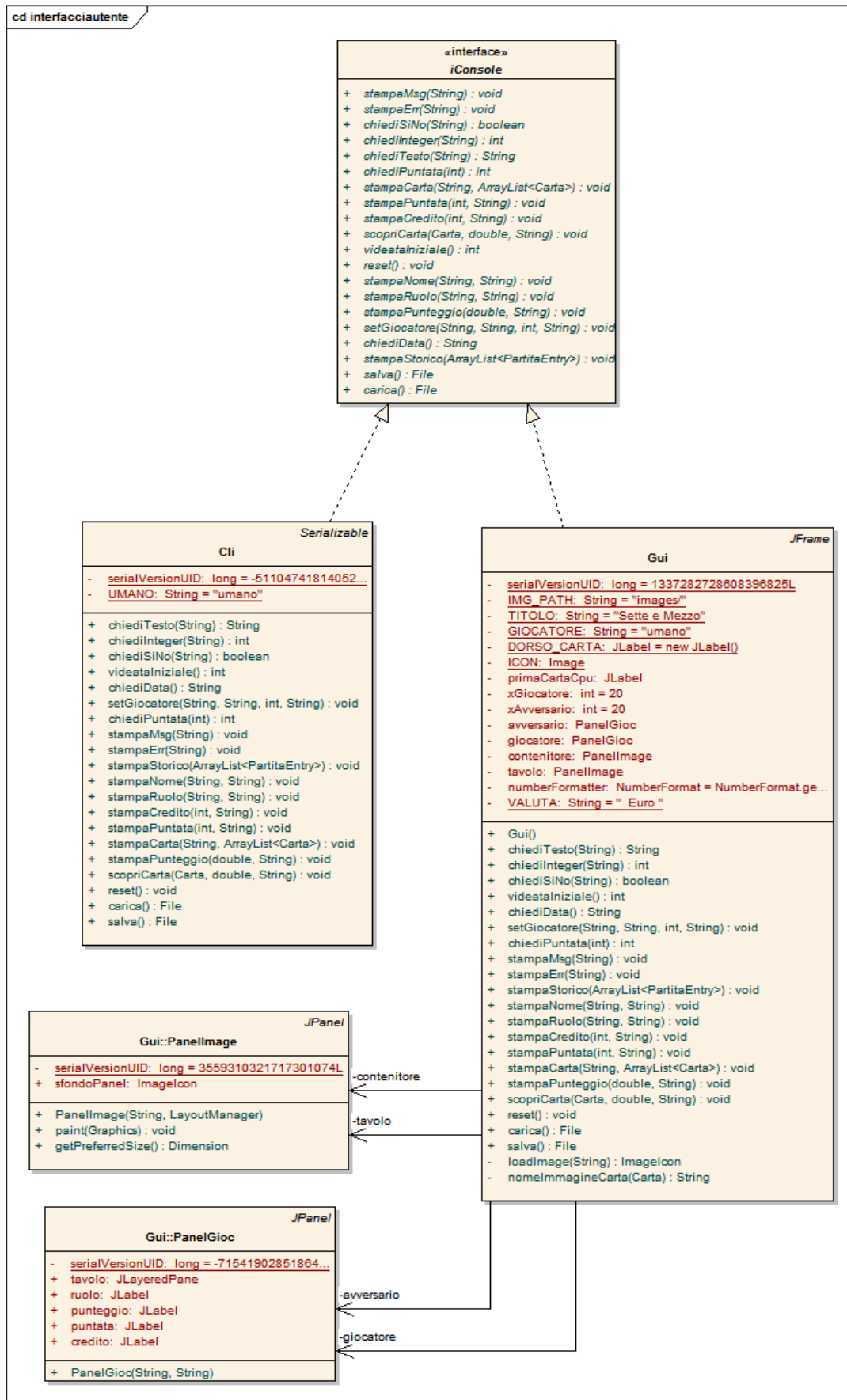
I casi evidenziati in rosso sono casi in cui lo sfidante vince la mano corrente e diventa banco alla mano successiva.

#### 6.2.6. settemezzo.interfacciautente

Il package raggruppa tutte le classi che sono necessarie all'interazione, testuale e grafica, del giocatore con il programma.

Rispetto alla soluzione proposta, sono state spostate nel sottopackage "util" le classi che effettuano la verifica dei dati di input, ed è stata rinominata la classe relativa all'interazione testuale. Il nuovo nome (*Cli*, Command line interface) risulta essere più appropriato e in linea con il nome utilizzato per la classe che gestisce l'interazione grafica (*Gui*, Graphical user interface).

I relativi sottopackage non sono previsti nella specifica base.



#### 6.2.6.1. iConsole

L'interfaccia *iConsole* ha il ruolo di definire i metodi necessari a consentire l'interazione tra l'utente e il gioco del Sette e Mezzo. Grazie ad essa si possono avere molteplici modalità d'interazione, lasciando inalterata la logica di gioco.

Rispetto alla specifica base, la classe ha mantenuto i metodi *stampaMsg()*, *chiediSiNo()*, *chiediInteger()*, *chiediTesto()* ed *videataIniziale()*. Diverso il discorso per i metodi *iniziaMano()* e *videataMano()* che sono stati eliminati e sostituiti da una serie di metodi che si sono dimostrati più adatti alla logica di gioco implementata.

Inoltre nella classe sono presenti due metodi per il salvataggio e il caricamento della partita; il metodo *carica()* restituisce un oggetto di tipo *File* che rappresenta il file di salvataggio da caricare. Il metodo *salva()* restituisce un oggetto di tipo *File* che rappresenta il file in cui salvare la partita. Questi due metodi sono stati aggiunti in quanto le modalità di richiesta del nome del file di salvataggio e della sua posizione, dipendono dall'interazione usata.

#### 6.2.6.2. Cli

La classe *Cli* è una delle due classi che si occupa dell'implementazione dei metodi definiti dall'interfaccia *iConsole*. Il ruolo di *Cli* è di implementare l'interazione tra l'utente e il gioco via testuale: questo consente all'utente di interagire con l'applicazione inviando comandi tramite tastiera e ricevendo risposte alle elaborazioni tramite testo scritto. Per la realizzazione dei metodi che richiedono un input da parte dell'utente, si sfrutta la classe *Keyboard*, la quale facilita tale operazione.

Per quanto riguarda l'implementazione dei metodi *carica()* e *salva()*, si è pensato di considerare come cartella di salvataggio quella in cui sia avvia il programma. File di salvataggio dislocate in altre cartelle del sistema, non potranno essere utilizzati: tale scelta è stata presa per facilitare la gestione del caricamento e del salvataggio da parte dell'utente, che così non deve preoccuparsi di inserire nessun percorso.

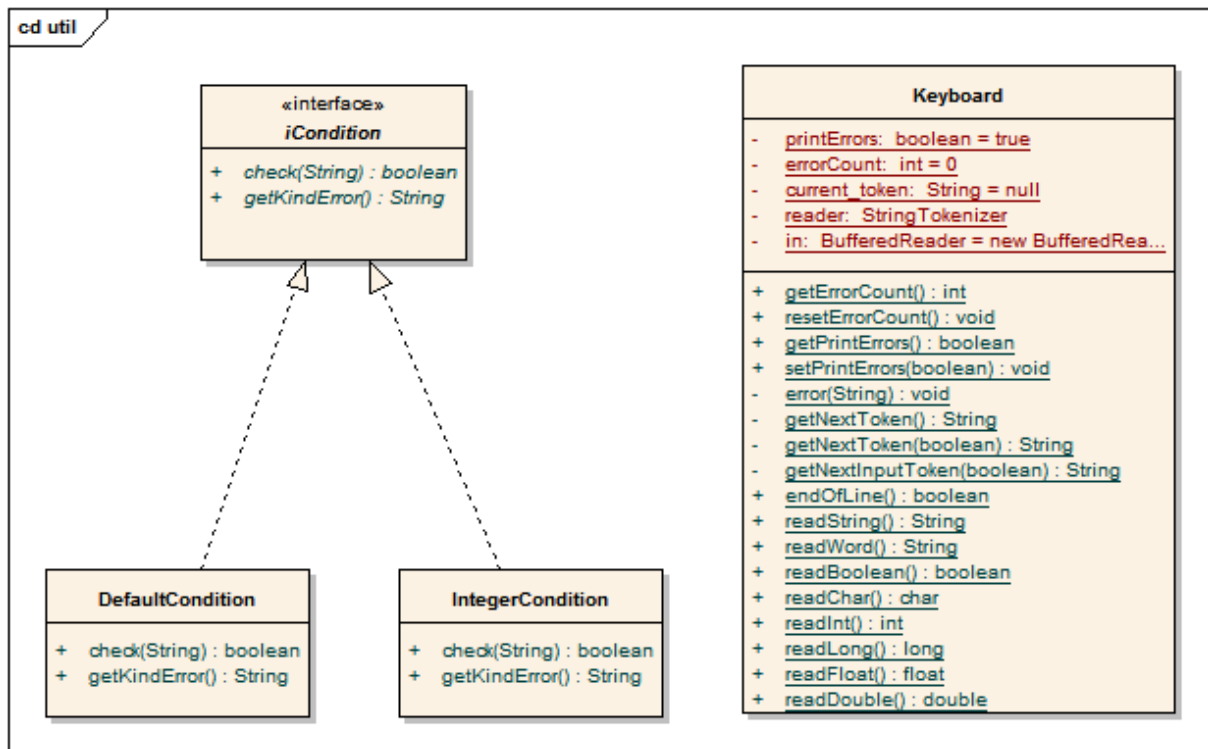
#### 6.2.6.3. Gui

*Gui* si presenta come la seconda classe che implementa i metodi definiti nell'interfaccia di *iConsole*. Il suo ruolo è di consentire all'utente di interagire con l'applicazione mediante finestre, icone, menu e dispositivo di puntamento (interazione WIMP). Grazie ad essa, dotiamo il gioco di una modalità grafica alternativa alla modalità testuale fornitaci dalla classe *Cli*.

Cambiando il metodo di interazione, cambia anche l'implementazione dei metodi. Per richiedere delle informazioni all'utente, la classe sfrutta i metodi offerti dalla classe *InputDialog*. Inoltre l'implementazione dei metodi *carica()* e *salva()* è stata migliorata rispetto alla logica presente in *Cli*. Si è deciso di far apparire, all'atto del salvataggio e del caricamento, una finestra di selezione file che permetta all'utente di salvare o caricare un file di salvataggio presente in qualsiasi cartella del computer.

#### 6.2.7. settemezzo.interfacciautente.util

Il package ha il compito di riunire tutte le classi che forniscono un ruolo di utility nell'interazione con l'utente. Presenta l'interfaccia *iCondition*, che definisce i metodi per controllare la validità di un input di una finestra di dialogo, e la classe che implementa tali metodi per la validazione di una stringa (*DefaultCondition*). A queste classi bisogna aggiungere la classe *Keyboard* utilizzata per la lettura dei dati inseriti da tastiera.



#### 6.2.7.1. iCondition

L'interfaccia *iCondition* ha il compito di definire i metodi per la validazione di un input di una Dialog.

L'interfaccia si presenta composta dai seguenti metodi:

- *check()*, che si occupa della validazione dell'input.
- *getKindError()*, che restituisce l'errore riscontrato durante una validazione errata.



#### 6.2.7.2. DefaultCondition

Tale classe implementa l'interfaccia *iCondition* e si occupa della realizzazione della condizione in base alla quale viene verificato se un testo è identificabile come una serie di caratteri non nulli.

L'implementazione del metodo *check()*, consiste nel verificare se la stringa è nulla oppure composta da soli spazi. In caso affermativo, viene restituito il valore falso altrimenti vero. Col metodo *getKindError()*, invece, ritorniamo una stringa che notifica l'inserimento di caratteri errati.

Un'ulteriore classe che implementa l'interfaccia *iCondition* è *IntegerCondition*. Prevista nella specifica base ed inizialmente implementata, è stata esclusa dal progetto in quanto, a seguito di modifiche, si dimostrava inutilizzata. Riportiamo qui il codice solo a scopo illustrativo.

```
public class IntegerCondition implements iCondition {

    @Override
    public boolean check(String input){
        try{
            Integer.parseInt(input);
            return true;
        }
        catch (NumberFormatException e){
            return false;
        }
    }

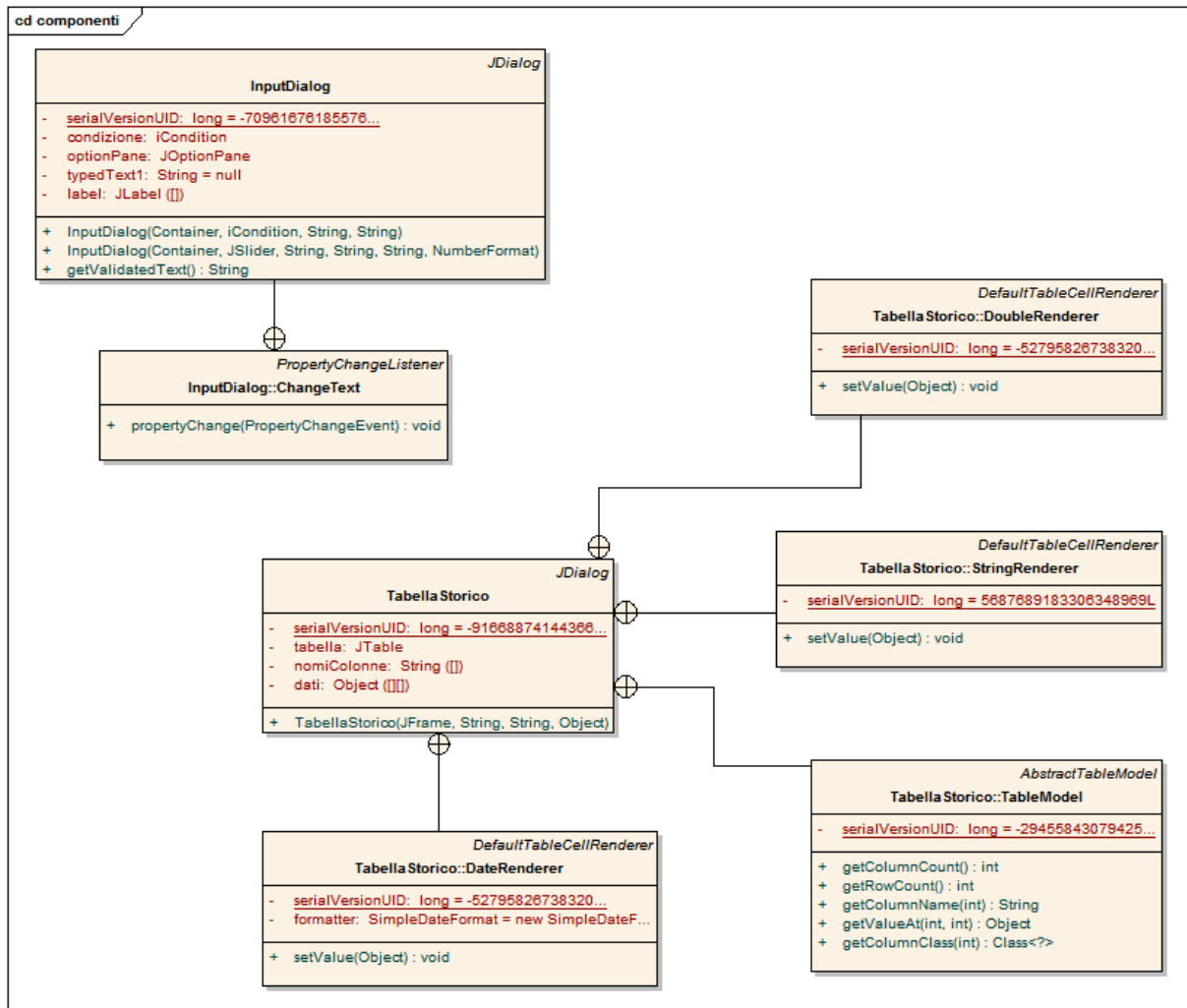
    @Override
    public String getKindError(){
        return "Hai inserito caratteri non numerici!";
    }
}
```

#### 6.2.7.3. Keyboard

Fornitaci assieme alla specifica iniziale, questa classe facilita la gestione dell'input da tastiera, creando un'astrazione sui dettagli riguardanti il parsing, la conversione e l'eventuale cattura di eccezioni sui dati di input.

### 6.2.8. settemezzo.interfacciautente.componenti

Il package raggruppa le classi che sono componenti grafiche di ausilio per l'interfaccia grafica: svolgono la funzione di acquisizione di input dall'utente e visualizzazione di informazioni riguardanti le partite giocate. Le classi che svolgono tale funzione sono *TabellaStorico* e *InputDialog*.



#### 6.2.8.1. InputDialog

La classe *InputDialog* ha la finalità di raccogliere le finestre modali di input utilizzate nel relativo prodotto. Secondo la specifica base, la classe doveva essere usata per presentare finestre di input per l'inserimento di dati di tipo intero o testuale. I dati inseriti venivano successivamente validati con una delle classi che implementa l'interfaccia *iCondition*. La classe realizzata non solo rispetta questa specifica ma la amplia: è possibile utilizzarla per permettere anche la visualizzazione di finestre di dialogo usate per il login o per mostrare una lista di opzioni.

Non tutte le tipologie di finestre vengono utilizzate in entrambi i prodotti; per questa ragione esistono due versioni di *InputDialog*, una per ogni prodotto, che presentano in ognuno le realizzazioni delle sole finestre di input usate nel prodotto.

Metodi	Prodotti		
	Stand Alone	Client / Server	
		Lato client	Lato server
InputDialog() input testuale	✓	✓	
InputDialog() input numerico	✓	✓	
InputDialog() input testuale doppio		✓	
InputDialog() input lista		✓	
getValidatedText()	✓	✓	
getValidatedTexts()		✓	

Nella tabella è possibile notare come le tipologie di finestre di input siano state suddivise; ognuna di queste finestre può essere visualizzata utilizzando il costruttore adatto.

La finestra di input testuale presenta una finestra di dialogo in cui è possibile inserire dei dati testuali, i quali saranno validati con la condizione passata come parametro. Stesso discorso vale anche per la finestra con un doppio input utilizzata, nel nostro caso, come finestra di login. Con la finestra per input numerico, viene mostrato uno slider presentato un minimo ed un massimo passati come parametri. Nella finestra di input con lista, viene mostrata una lista di opzioni da cui selezionarne una; nel nostro progetto è stata usata per far scegliere all'utente quale salvataggio caricare.

I metodi *getValidatedText()* ed *getValidatedTexts()* servono per recuperare l'input inserito dopo la validazione rispettivamente da una finestra con singolo e doppio input testuale.

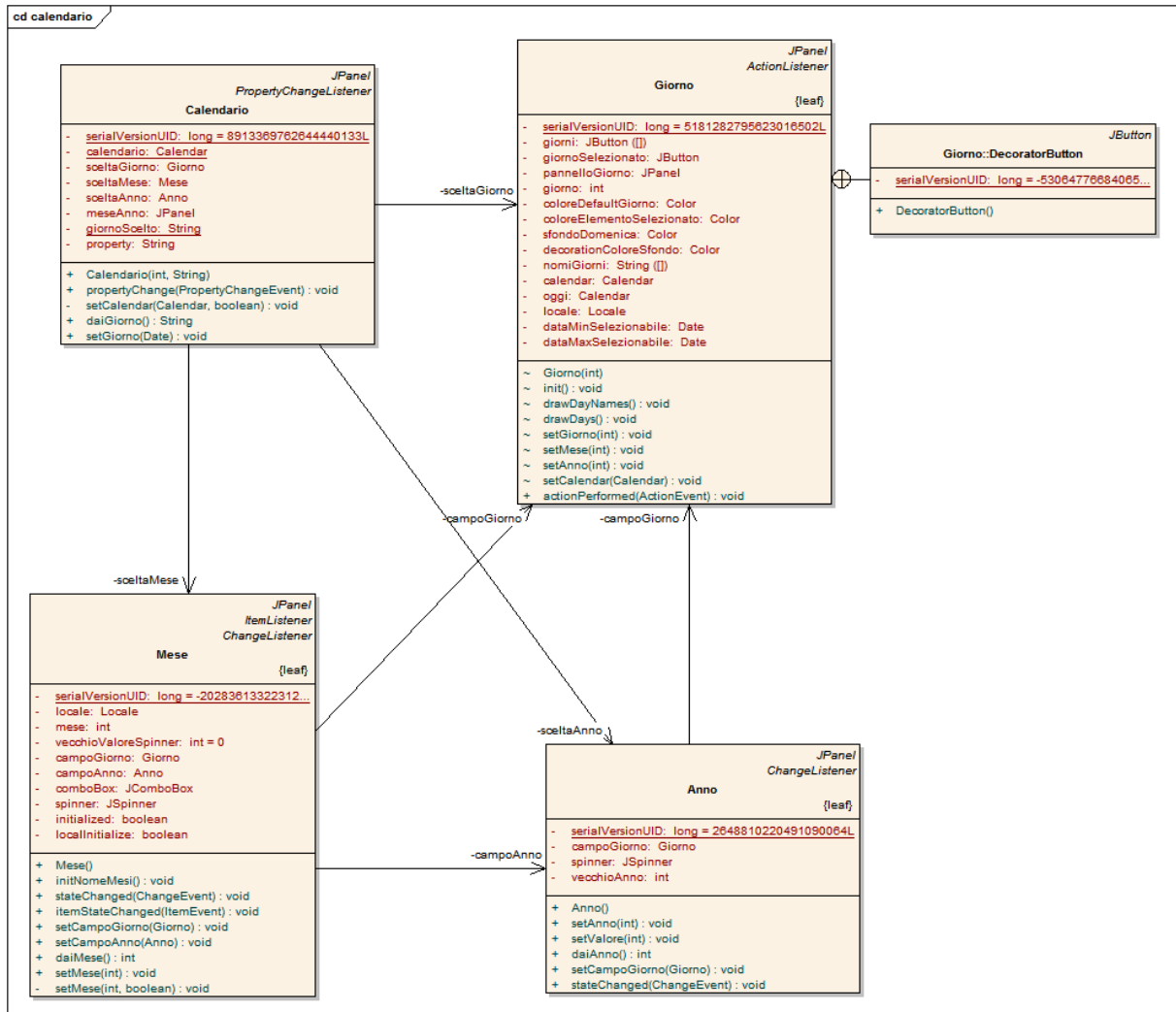
#### 6.2.8.2. **TabellaStorico**

*TabellaStorico* viene utilizzata per la visualizzazione delle partite giocate sotto forma tabellare. Per aumentare la riusabilità, si è deciso di consentire l'inserimento dei nomi delle colonne assieme ai dati da visualizzare, in modo tale da permettere la visualizzazione di qualsiasi informazione che si ritenga più leggibile sotto forma tabellare.

La tabella presenta anche dei formattatori per le stringhe, per i dati di tipo Double e per le date. Le stringhe vengono visualizzate in ogni cella con un allineamento orizzontale. I dati di tipo Double sono anch'essi visualizzati con allineamento orizzontale e con la virgola, anche se non presenta parte decimale. Le date, invece, sono formattate nel seguente formato "gg-mm-aaaa - hh:mm:ss".

### 6.2.9. settemezzo.interfacciautente.componenti.calendario

Raccoglie tutte le classi che sono necessarie alla visualizzazione grafica di un calendario in cui selezionare un giorno passato, futuro o del mese corrente. Nella versione Client/Server il suo nome è stato modificato in “settemezzo.applet.componenti.calendario” ma le classi in esso contenute sono rimaste invariate.



#### 6.2.9.1. Calendario

La classe *Calendario* costruisce un componente grafico che riproduce le funzioni di un calendario.

Presenta un pannello nel quale appaiono due campi per la selezione del mese e dell'anno, e una matrice di bottoni per rappresentare i giorni del mese selezionato. Alla prima apparizione del pannello, la data selezionata è quella che il sistema ricava dal sistema operativo. In seguito alla scelta della data per la quale sono desiderate delle informazioni, la classe restituisce il controllo al metodo chiamante (il metodo *chiediData()* della classe *Gui* per la versione Stand Alone e *MainApplet* per la versione Client/Server) che è incaricato di gestire l'evento.

La classe *Calendario* è l'unica classe che nel package ha visibilità pubblica, ed utilizza le funzionalità delle classi *Anno*, *Mese* e *Giorno* che invece hanno visibilità di package.

Per la presentazione della data *Calendario* utilizza la classe *Calendar* della libreria “java.util”.

#### 6.2.9.2. Anno

La classe *Anno* costruisce un pannello con una casella di testo e due piccoli pulsanti sul lato. Quando si clicca sui pulsanti, l'anno viene incrementato o decrementato.

È contenuta nel package "calendario" ed utilizza la classe *Calendar* della libreria "java.util" per ricavare l'anno selezionato.

#### 6.2.9.3. Mese

La classe *Mese* costruisce un pannello con una casella di testo. E' possibile incrementare o decrementare il valore di quest'ultima o cliccando su due piccoli pulsanti sul lato oppure selezionando una scelta da un menù a tendina.

È contenuta nel package "calendario" ed utilizza la classe *Calendar* della libreria "java.util" per ricavare il mese selezionato.

#### 6.2.9.4. Giorno

La classe *Giorno* costruisce un pannello nel quale appare una matrice di bottoni i primi sette rappresentano i nomi dei giorni della settimana abbreviati secondo la localizzazione italiana, i restanti rappresentano i giorni del mese scelto.

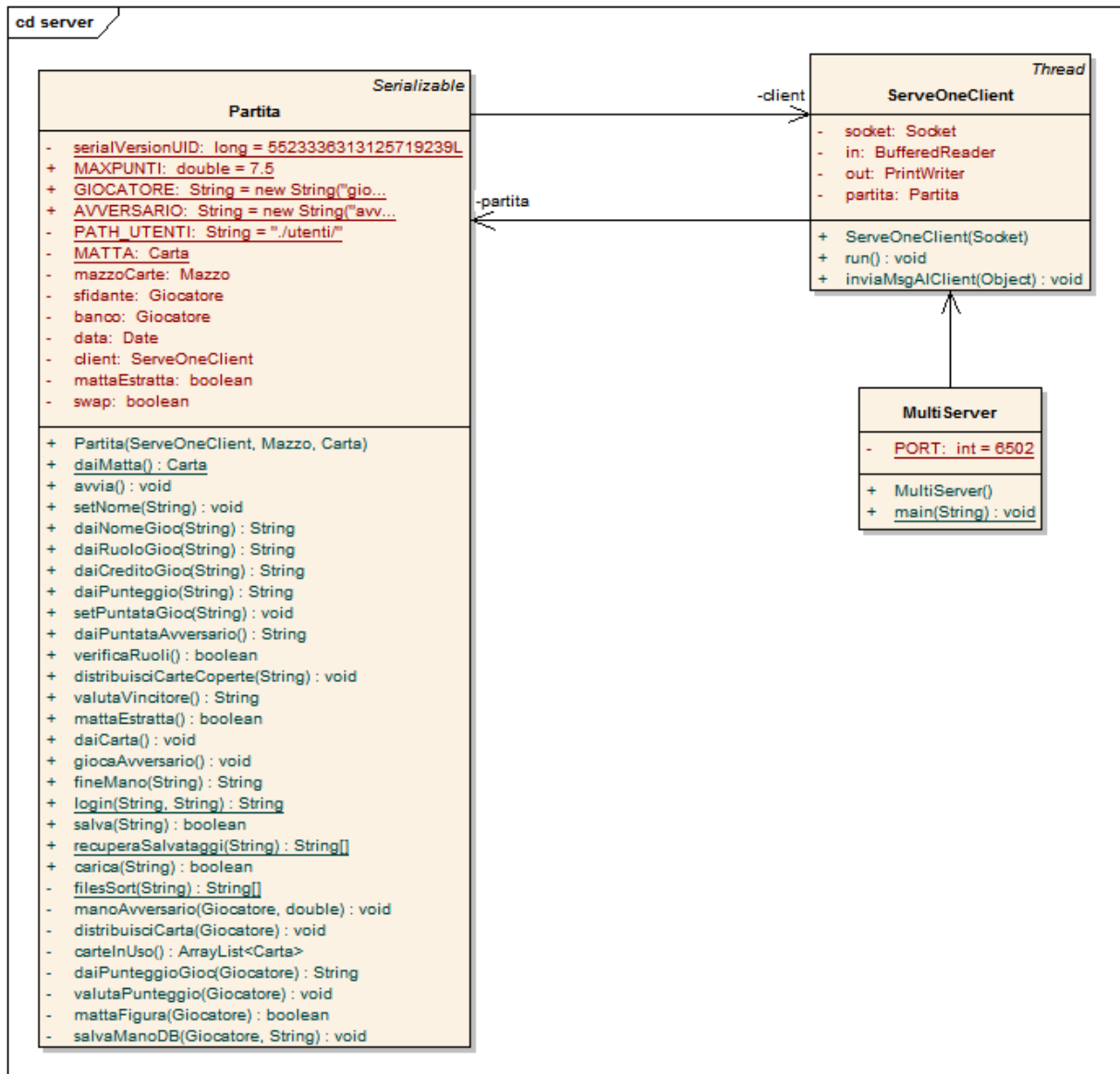
Cliccando un numero prefissato di volte lo stesso pulsante, è possibile sollevare un evento per la data che quel bottone rappresenta.

È contenuta nel package "calendario" ed utilizza:

- la classe *Calendar* della libreria "java.util" per ricavare il giorno, il mese e l'anno selezionato;
- la classe interna *DecorationButton* utile per migliorare l'aspetto grafico dei bottoni.

### 6.2.10. settemezzo.server

Presente anche nel progetto base, il package “settemezzo.server” contiene le classi per la gestione del server. Rispetto al progetto base, però, le classi che lo compongono sono *MultiServer*, *ServeOneClient* e *Partita*. Questa decisione è stata presa in quanto è stato ritenuto più adatto lasciare le classi *Mazzo* e *MazzoTerminatoException* nei rispettivi package, per favorire una più facile manutenibilità e il riuso del codice.



#### 6.2.10.1. MultiServer

La classe *MultiServer* svolge la funzione di server multi-threading, ossia accetta le richieste di molteplici client consentendo l'esecuzione contemporanea di molteplici partite a Sette e Mezzo.

La sua realizzazione si presenta molto semplice; all'avvio la classe istanzia una socket ed occupa la porta “6502” del sistema operativo in uso, rimanendo in ascolto di un client. Ogni qual volta un client stabilisce una connessione, il server istanzia la classe *ServeOneClient* la quale si occupa dell'interazione tra il client e la logica del gioco residente sul server. Dopo l'istanziatura ritorna nello stato di ascolto e attende un'altra richiesta.

### 6.2.10.2. ServeOneClient

*ServeOneClient* è la classe principale del server: infatti essa si occupa della diretta interazione tra il client, il suo avversario e la logica di gioco. Per consentire tale interazione, la classe sfrutta il multithreading, permettendo a più client di potersi connettere contemporaneamente al server.

Per permettere l'interazione tra le due parti, al momento della connessione vengono instaurati due canali di comunicazioni, rispettivamente uno in ingresso ed uno in uscita; questi canali consentiranno lo scambio di messaggi tra il client e il server.

Dopo aver instaurati i canali di comunicazioni, la classe rimane in attesa di una richiesta da parte del client; richiesta che verrà successivamente elaborata dalla logica della partita. Per permettere l'interazione tra client e server è stato ideato un semplice protocollo di comunicazione composto da parole chiave ed eventuali dati necessari al completamento della richiesta.

Esaminiamo ora il protocollo creato:

- **start**&<nomeGioc>: con questa richiesta si chiede al server di iniziare una nuova partita e di impostare il nome dell'utente con il dato passato assieme alla richiesta.
- **setNomeGioc**&<nomeGioc>: grazie a questa richiesta, viene impostato il nome dell'utente.
- **nomeGioc**&<tipoGioc>: il client richiede il nome di un giocatore; se <tipoGioc> è uguale alla stringa "Giocatore" verrà fornito il nome dell'utente, altrimenti il nome del giocatore gestito dal computer.
- **ruoloGioc**&<tipoGioc>: il client richiede il ruolo di un giocatore; se <tipoGioc> è uguale alla stringa "Giocatore" verrà fornito il ruolo dell'utente, altrimenti il ruolo del giocatore gestito dal computer.
- **iniziaMano**: il client comunica al server la possibilità di iniziare una nuova mano, che porta alla verifica di un possibile scambio dei ruoli tra i giocatori. Il server risponde inviando il messaggio "true" o "false" rispettivamente se sia necessario effettuare uno scambio dei ruoli o no.
- **mattaEstratta**: richiesta che porta alla verifica della eventuale estrazione della carta identificata come matta. Il server risponde con il messaggio "true" o "false" rispettivamente se la matta è stata estratta o meno.
- **cartaCoperta**&<giocatore>: il client richiede la distribuzione della carta coperta per il giocatore indicato; se <giocatore> è pari alla stringa "sfidante" verrà distribuita la carta coperta allo sfidante, altrimenti al banco. Il server risponde con l'invio di una stringa con i dati sulla carta distribuita seconda una struttura chiave – valore.
- **creditoGioc**&<tipoGioc>: il client effettua una richiesta di recupero del credito del giocatore in esame. Se <tipoGioc> è uguale alla stringa "Giocatore" verrà fornito il credito residuo dell'utente, altrimenti il credito del giocatore gestito dal computer.
- **invioPuntata**&<puntata>: richiesta di impostazione della puntata per lo sfidante.
- **puntataAvversario**: il client richiede la puntata del giocatore che assume il ruolo di sfidante.
- **altraCarta**: il client simula la richiesta da parte dell'utente di una nuova carta; il server elabora la richiesta e invia al client una stringa con i dati sulla carta distribuita seconda una struttura chiave – valore.
- **punteggioGioc**&<tipoGioc>: il client richiede il punteggio totalizzato dal giocatore passato assieme alla richiesta. Il server invia al client il punteggio totalizzato e lo stato del giocatore, ovvero se ha sballato, totalizzato sette e mezzo oppure se è ancora in gioco. Tutte le informazioni sono contenute in una stringa secondo una struttura chiave – valore.
- **stopCarte**: il client comunica al server che l'utente ha scelto di "stare" con le carte in possesso, lasciando il turno al giocatore avversario.



- **valutaVincitore:** il client richiede al server di determinare il vincitore. Il server risponde con un messaggio preparato con il nome del giocatore vincitore.
- **resetta**&<emailGioc>: il client comunica al server di finalizzare la mano. Il server esegue le operazioni di finalizzazione e comunica al client se ci sono le condizioni per poter continuare una partita.
- **login**&<emailGioc>&<pswGioc>: il client richiede di autenticare il giocatore con i dati inviati assieme alla richiesta. Se l'autenticazione è andata a buon fine, il server ritorna il nickname usato dal giocatore, altrimenti ritorna una stringa nulla.
- **salvaPartita**&<emailGioc>: il client invia la richiesta di salvataggio della partita corrente per il giocatore che si è registrato con la email invia assieme alla richiesta. Dopo il salvataggio in una cartella che presenta lo stesso nome dell'email del giocatore, il server ritorna "true" se l'operazione è andata a buon fine, "false" altrimenti.
- **recuperaSalvataggi**&<emailGioc>: il client richiede il recupero di tutti i nomi dei file relativi ai salvataggi del giocatore che si è registrato con l'email inviata assieme alla richiesta. Inoltre, invia anche una parola chiave di fine dell'invio dei nomi dei file (**stop**).
- **caricaPartita**&<nomeFile>: il client richiede al server di caricare la partita presente nel file di salvataggio con il nome inviato assieme alla richiesta. Il server, dopo aver elaborato la richiesta, risponde con l'esito del caricamento.
- **finePartita:** il client comunica al server la terminazione della sessione di gioco.

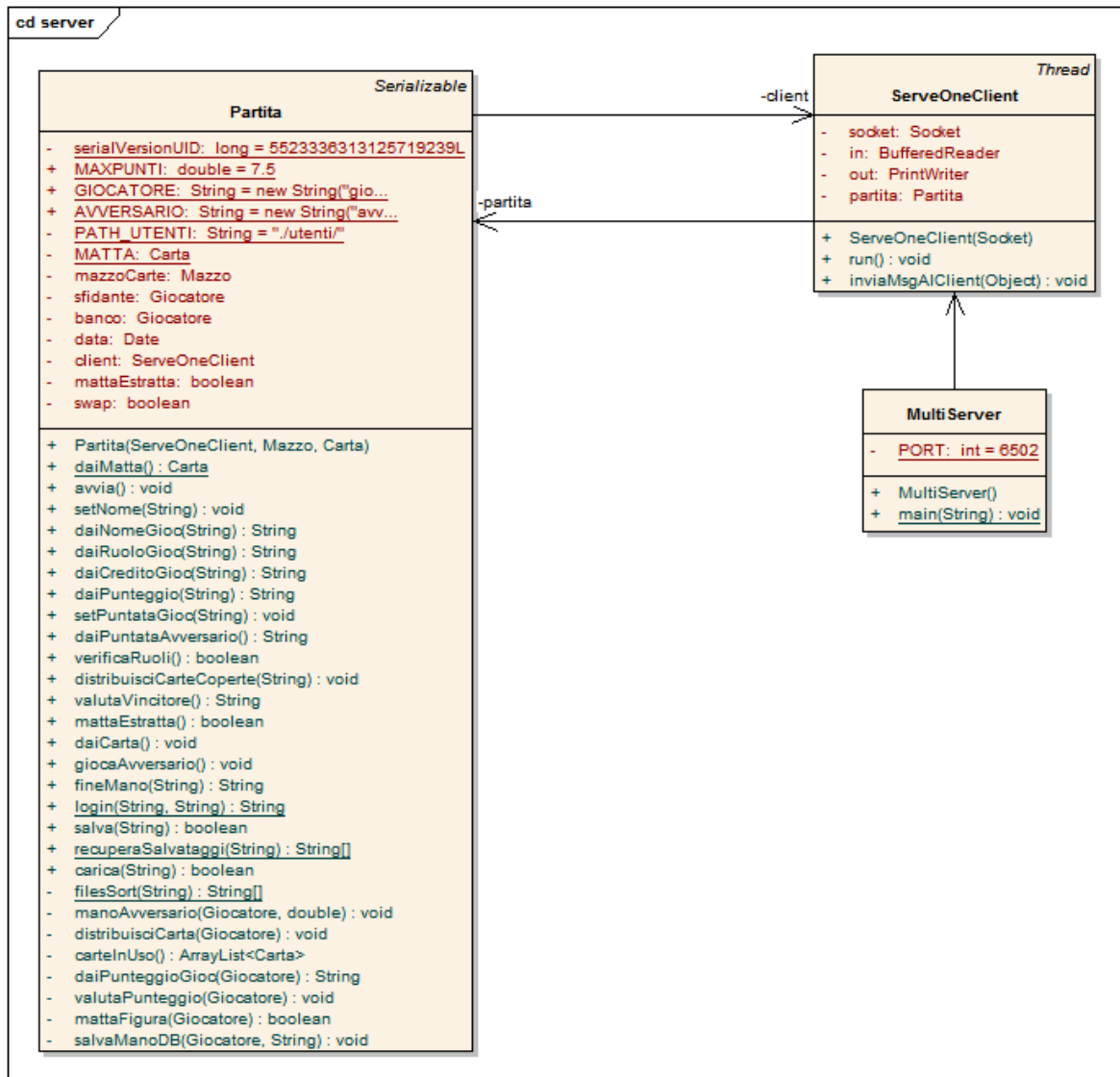
Come è possibile notare, in alcune richieste il server risponde con una stringa strutturata secondo una logica chiave – valore; tale particolarità è stata adottata per:

- evitare il moltiplicarsi dei messaggi scambiati.
- consentire un recupero più semplice ed organizzato dei dati: grazie alla struttura chiave – valore si è sempre a conoscenza della semantica dell'informazione inviata.

Altra particolarità adottata, è l'invio delle richieste al server concatenate con i dati: per recuperare le richieste, separandole dai dati, il server "splitta" la stringa secondo il carattere &. Tale soluzione è stata adottata per ridurre ancora una volta il numero di messaggi scambiati tra client e server.

### 6.2.10.3. Partita

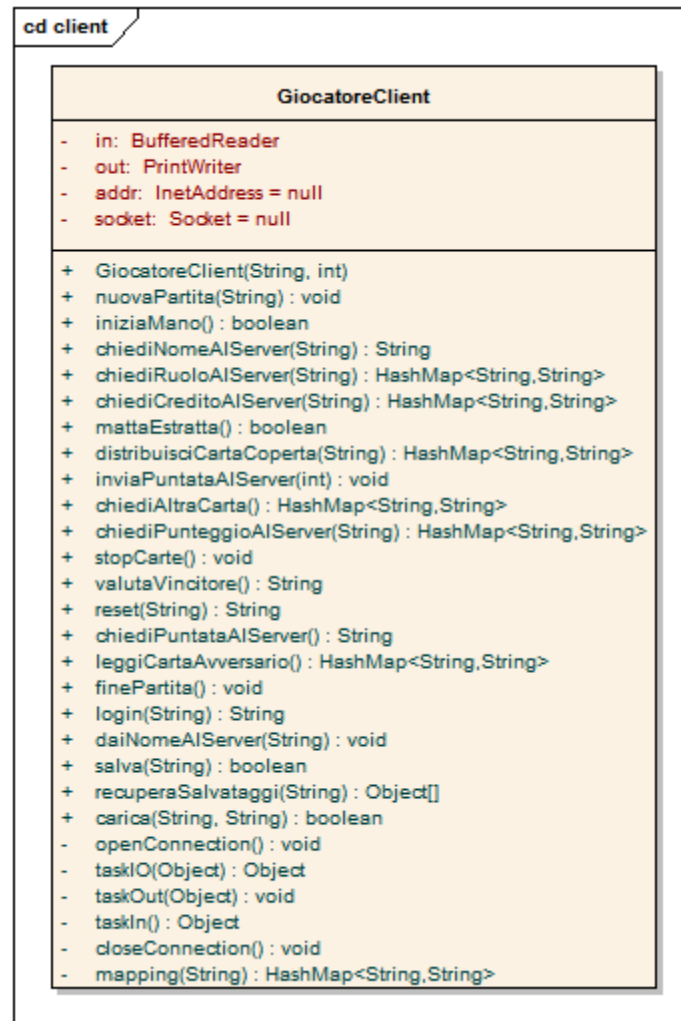
Evoluzione della classe partita descritta riguardo il package "settemezzo".



Le uniche differenze rispetto a quest'ultima sono l'aggiunta del campo *swap*, che indica la necessità di effettuare uno scambio dei ruoli, e il nome del file binario che adesso presenta un preciso formato: "gg-mm-aaaa hh.mm.ss.dat" ossia il giorno e l'ora in cui viene eseguito il salvataggio della partita. Per quanto riguarda il caricamento, il metodo non subisce cambiamenti. Per finire, nella classe *Partita* in questione, è stato previsto il metodo *salvaManoDB()* che salva la mano appena conclusa nel relativo database del prodotto. Le implementazioni differiscono dal fatto che nella versione Stand Alone vengono salvate le mani concluse di qualsiasi giocatore indipendentemente dal ruolo, mentre nella versione Client/Server solo le mani concluse dei giocatori registrati.

### 6.2.11. settemezzo.client

Presente nella specifica base, il package “*settemezzo.client*” contiene le classi per la gestione del client. Si presenta composta dalla sola classe *GiocatoreClient* che si occupa dell’interazione tra client e server. Si è ritenuto più giusto non spostare la classe *Partita* e la classe *Carta* nel package in esame, in quanto deve essere il server ad occuparsi della gestione della partita e delle sue regole, del calcolo del punteggio e della distribuzione delle carte ai giocatori. Il ruolo del client deve essere quello di visualizzare le carte e di permettere l’interazione con la logica di gioco residente sul server.



#### 6.2.11.1. **GiocatoreClient**

*GiocatoreClient* svolge la funzione di gestore dei messaggi tra il client e il server. Grazie a questa classe, il client è capace di comunicare con il server inviando messaggi necessari per l'avvio, l'esecuzione e la terminazione del gioco.

Affinché la comunicazione sia possibile, nel costruttore della classe sono presenti le istruzioni che consentono la creazione del flusso di comunicazione con il server. Inizialmente viene creato un oggetto *InetAddress* che rappresenta l'indirizzo IP del server; successivamente viene creata una socket ed instaurati i canali di comunicazione in ingresso ed in uscita.

Oltre al costruttore, sono presenti anche i metodi necessari alla comunicazione con il server: per ogni parola chiave prevista nel protocollo di comunicazione, è stato creato un relativo metodo che si occupa d'inviare la richiesta e di recuperare la risposta. Per consentire l'utilizzo della risposta inviata dal server, le sole stringhe ricevute sotto forma di chiave – valore vengono elaborate e trasformate in un *HashMap*: è stata scelta tale struttura in quanto si presenta strutturata in coppie chiave – valore, facilitando la conversione e l'utilizzo dei dati.

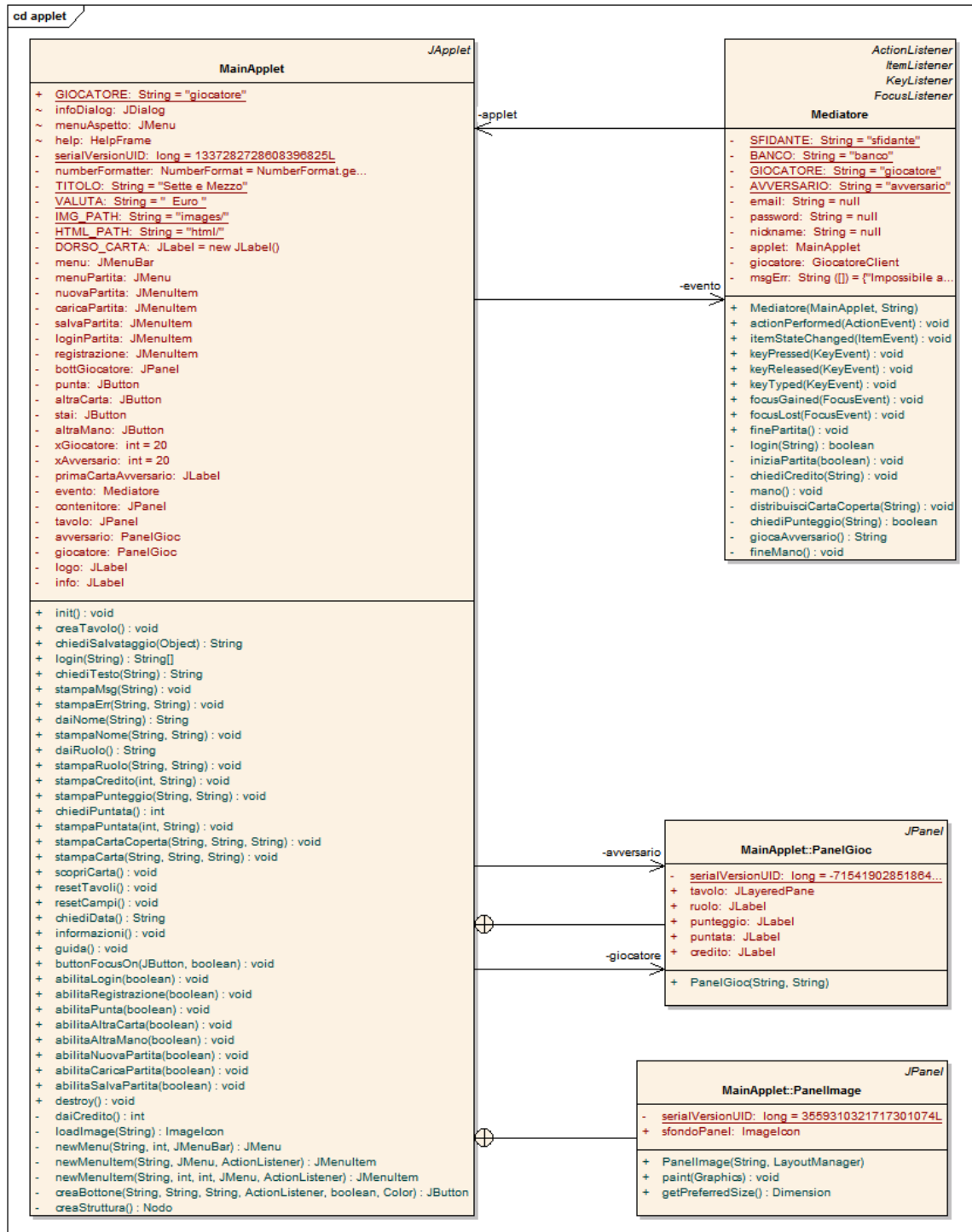
Le altre risposte del server non vengono in alcun modo modificate ma direttamente utilizzate.

### 6.2.12. settemezzo.applet

Contiene le classi per l'esecuzione del client tramite Applet. Oltre alla classe che implementa l'applet del gioco lato client (*MainApplet*), è presente anche la classe *Mediatore* che si occupa di gestire gli eventi scatenati dall'interfaccia utente.

Rispetto al progetto base, si può notare l'aggiunta della classe *Mediatore*, che si occupa della gestione degli eventi. I sottopackage presenti in esso non sono previsti dal progetto base.

#### 6.2.12.1. MainApplet



*MainApplet* è la classe principale del prodotto lato Client, ossia rimpiazza le classi *MainClass* e *Gui* del prodotto Stand Alone. La classe in questione è una evoluzione della classe *Gui* presente nel prodotto Stand Alone. Come *Gui*, *MainApplet* svolge il ruolo di interfaccia utente grafica, consentendo all'utente di interagire con l'applicazione mediante finestre, icone, menu e dispositivo di puntamento (interazione WIMP). La progettazione dell'interfaccia grafica è stata operata tenendo conto delle principali linee guida di usabilità definite dall'ISO 9241.

A differenza di *Gui*, questa classe estende la classe *JApplet* e quindi utilizzabile incorporata in una pagina html. La classe ha il compito di visualizzare le informazioni relative ai due giocatori, le carte in possesso di ogni giocatore e visualizzare messaggi di notifica sullo stato della partita.

La struttura grafica della classe è molto semplice; possiamo dividere l'intera applet in tre parti: la barra dei menù, il tavolo da gioco, i pulsanti d'interazione. Per una descrizione dettagliata di queste parti, si rimanda alla guida in linea presente nell'applicazione.

I metodi presenti nella classe si occupano della visualizzazioni delle informazioni dei singoli utenti, della visualizzazione delle carte.

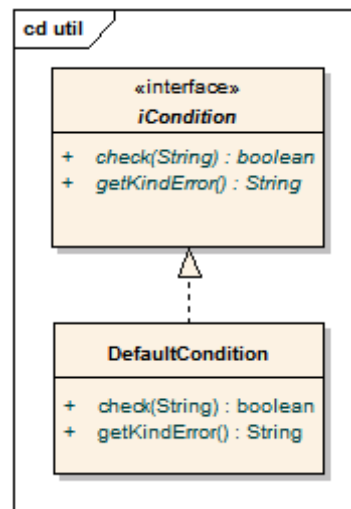
#### **6.2.12.2. Mediatore**

La classe *Mediatore* si presenta come classe di ausilio nella gestione dell'interfaccia grafica fornita da *MainApplet*. Il suo ruolo è quello di gestire gli eventi scatenati dalle componenti attive della classe *MainApplet*. La scelta di inserire questa classe è dovuta alla volontà di separare la parte grafica dalla logica di gestione di essa.

Quando una componente attiva viene attivata dall'utente, la classe *Mediatore* individua la componente selezionata ed esegue le opportune operazioni. Molte componenti attive richiedono un'interazione con il server e, di conseguenza, chiamate ai metodi di *GiocatoreClient*. Le componenti che richiedono l'interazione con il server sono i pulsanti di interazione, le voci del menù riguardante il login, la registrazione, il salvataggio e il caricamento di una partita.

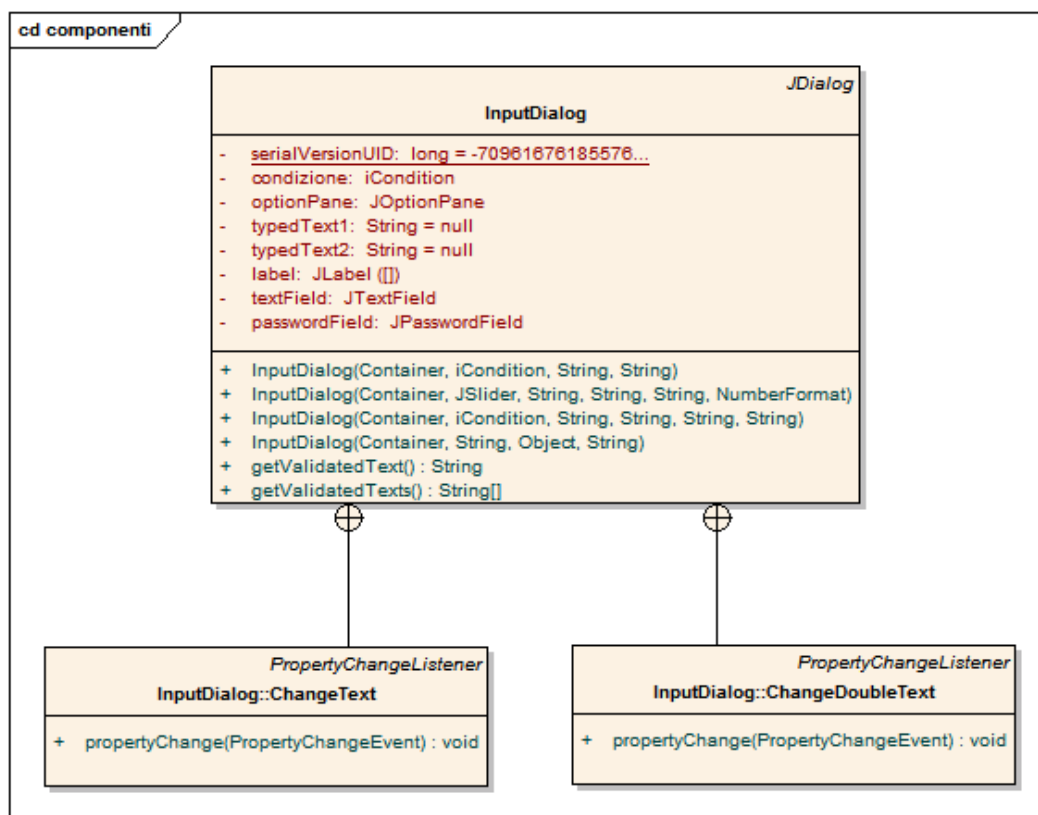
### 6.2.13. `settemezzo.applet.util`

Creato sulla base del package “`settemezzo.interfacciautente.util`”, differisce da esso per la mancanza della classe `Keyboard` non utilizzata nella versione Client/Server.



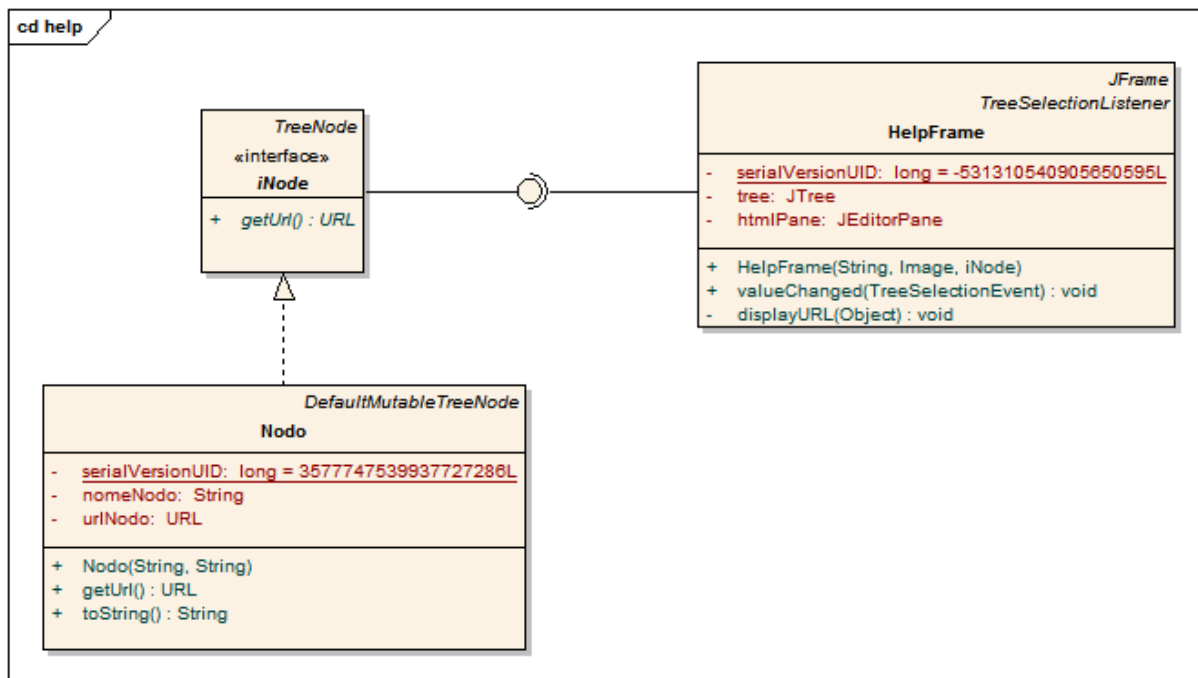
### 6.2.14. `settemezzo.applet.componenti`

Differisce dal package “`settemezzo.interfacciautente.componenti`” per la sola mancanza della classe `TabellaStorico`, poiché le informazioni relative allo storico delle partite sono reperibili mediante l’uso di pagine [JSP](#).



### 6.2.15. `settemezzo.applet.componenti.help`

Questo package raccoglie le classi necessarie alla visualizzazione di un help in linea utilizzabile dall'utente per capire il funzionamento del gioco. Le classi che compongono tale package sono *HelpFrame*, *iNode* e *Nodo*.



#### 6.2.15.1. *HelpFrame*

La classe permette la visualizzazione di una guida in linea, detta anche help. Tale classe può essere integrata in qualsiasi progetto ove si richieda la visualizzazione di un help che aiuti l'utente nell'utilizzo del software.

L'help si presenta costituito da un indice strutturato ad albero e da una parte di visualizzazione delle pagine di help. Tali pagine sono semplici file html contenenti le informazioni da visualizzare.

Per utilizzare questa classe è necessario fornire la radice della struttura ad albero che implementa l'indice dell'help: in tale struttura saranno inseriti i nomi dei file html da utilizzare.



#### 6.2.15.2. *iNode*

L'interfaccia *iNode* viene impiegata nella realizzazione della struttura ad albero usata come indice di un *HelpFrame*.

Estende l'interfaccia *TreeNode*, presente in Java e adoperata per definire i requisiti per un oggetto di poter essere usato come nodo per il componente grafico *JTree*; tale componente è usato nella realizzazione dell'indice della guida in linea. Estendendo l'interfaccia *TreeNode* si ereditano tutti i relativi metodi e ciò consente alla nostra classe di essere utilizzata in un *JTree*. *iNode*, inoltre, introduce il metodo *getUrl()* che restituisce l'url della pagina associata al nodo in esame. Grazie a questo metodo, *iNode* può essere utilizzato nella realizzazione di un indice per la guida in linea.

#### 6.2.15.3. *Nodo*

La classe implementa l'interfaccia *iNode* ed rappresenta un nodo utilizzabile in una struttura ad albero, in particolare per la definizione di indici per la classe *HelpFrame*. Come si può notare dal codice sorgente, *Nodo* estende la classe Java *DefaultMutableTreeNode*, la quale implementa i metodi definiti nell'interfaccia *TreeNode*: in questo modo la classe *Nodo* si occupa della sola implementazione del metodo *getUrl()*.

L'implementazione del metodo *getUrl()* risulta molto semplice: viene recuperato dal nodo il nome del file html associato, calcolato il percorso del file e restituito al chiamante.

### 6.3. Modello di decisione

La tabella riassume la suddivisione delle classi per prodotto, evidenziando le classi comuni e non.

Classi	Prodotti		
	Stand Alone	Client / Server	
		Lato client	Lato server
Giocatore	✓		✓
GiocatoreUmano	✓		✓
BancoCPU	✓		✓
SfidanteCPU	✓		✓
Mazzo	✓		✓
MazzoNapoletano	✓		✓
Carta	✓		✓
Seme	✓		✓
TipoCarta	✓		✓
MazzoTerminatoException	✓		✓
SballatoException	✓		✓
SetteMezzoException	✓		✓
Alias	✓		✓
DbAccess	✓		✓
iPartitaDAO	✓		✓
PartitaDAOFactory	✓		✓
PartitaEntry	✓		✓
TransazioneDB	✓		✓
UtenteEntry			✓
ClasseMain	✓		
Partita	✓		✓
iConsole	✓		
Cli	✓		
Gui	✓		
iCondition	✓	✓	

DefaultCondition	✓	✓	
Keyboard	✓		
InputDialog	✓	✓	
TabellaStorico	✓		
Calendario	✓	✓	
Anno	✓	✓	
Mese	✓	✓	
Giorno	✓	✓	
MultiServer			✓
ServeOneClient			✓
GiocatoreClient		✓	
MainApplet		✓	
Mediatore		✓	
HelpFrame		✓	
iNode		✓	
Nodo		✓	

## 6.4. Progettazione database

Mostriamo di seguito la progettazione del database in dettaglio relativo sia alla versione Stand Alone sia alla versione Client/Server. In quest'ultima versione il database serve oltre che per memorizzare le informazioni utili per la visualizzazione dello storico, anche per memorizzare i dati di accesso degli utenti.

### 6.4.1. Dettaglio dei dati

Mostriamo di seguito i dettagli circa i dati presenti nella tabella transazione del prodotto Stand Alone.

Identificatore campo	Chiave	Tipo	Descrizione
id_partita	Primaria	INT(11)	11 cifre intere usate per identificare univocamente la partita salvata.
giocatore	-	VARCHAR(20)	20 caratteri usati per memorizzare il nickname scelto dal giocatore
ruolo	-	VARCHAR(8)	8 caratteri usati per memorizzare il ruolo che il giocatore ricopre in una determinata mano
punteggio	-	DOUBLE	Punteggio totalizzato dal giocatore in una mano
vincita	-	VARCHAR(6)	6 caratteri per memorizzare il denaro vinto/perso in una mano
data	-	DATETIME	La data durante la quale si è svolta la mano

Mostriamo di seguito i dettagli circa i dati presenti nella tabella “utente” del prodotto Client/Server.

Identificatore campo	Chiave	Tipo	Descrizione
email	Primaria	VARCHAR(20)	20 caratteri usati per memorizzare l'email del giocatore
nickname	-	VARCHAR(20)	20 caratteri usati per memorizzare il nome utente del giocatore
password	-	VARCHAR(10)	10 caratteri usati per memorizzare la password del giocatore

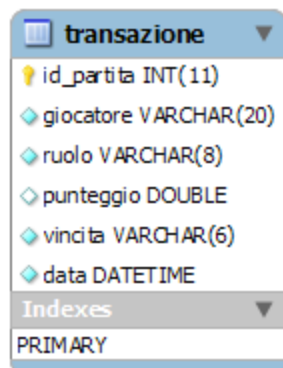
Mostriamo di seguito i dettagli circa i dati presenti nella tabella “transazione” del prodotto Client/Server.

Identificatore campo	Chiave	Tipo	Descrizione
id_partita	Primaria	INT(11)	11 cifre intere usate per identificare univocamente la partita salvata.
giocatore	Esterna	VARCHAR(20)	20 caratteri usati per memorizzare l'email del giocatore
ruolo	-	VARCHAR(8)	8 caratteri usati per memorizzare il ruolo che il giocatore ricopre in una determinata mano
punteggio	-	DOUBLE	Punteggio totalizzato dal giocatore in una mano
vincita	-	VARCHAR(6)	6 caratteri per memorizzare il denaro vinto/perso in una mano
data	-	DATETIME	La data durante la quale si è svolta la mano

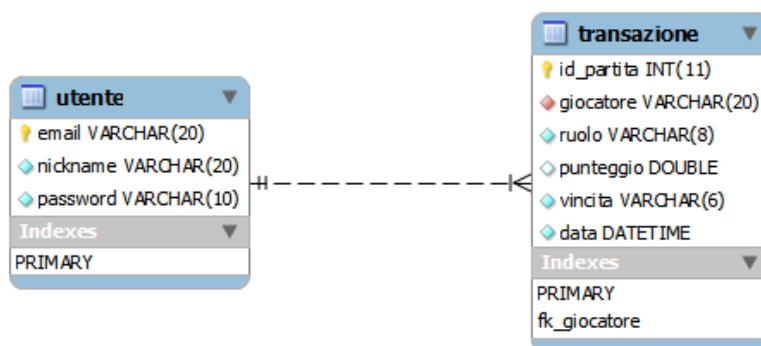
### 6.4.2. Modello del database

Presentiamo adesso i modelli grafici dei database sviluppati per il nostro progetto.

Il primo modello è relativo alla versione Stand Alone e comprende la sola tabella “*transazione*” che raccoglie le informazioni utili per visualizzare lo storico delle partite.



Il secondo è, invece, relativo alla versione Client/Server e comprende oltre alla tabella “*transazione*” anche la tabella “*utente*” che permette di salvare e verificare le credenziali di accesso degli utenti.



Come si evince dal modello, nella tabella “*transazione*” è presente una chiave esterna che pone un vincolo referenziale tra il campo “*giocatore*” e il campo “*email*” della tabella “*utente*”. Un utente può quindi registrarsi con una determinata email soltanto una volta. In fase di interrogazione del database circa lo storico, saranno recuperate in base all'email le partite dell'utente.

### 6.4.3.Script SQL

Dopo aver progettato il database, sono stati creati due script SQL per permettere la creazione dei suddetti database all'interno del DBMS MySQL. Negli script sono presenti le istruzioni necessarie alla creazione dei database sopra descritti.

Esaminando gli script, possiamo notare la presenza di istruzioni che portano alla creazione di un nuovo utente e alla assegnazione dei privilegi sul database creato. All'utente, chiamato user, sono concessi, tramite clausola GRANT, i privilegi di inserimento (INSERT) delle informazioni e recupero (SELECT) delle stesse. Non sono stati aggiunti ulteriori privilegi in quanto il recupero e l'inserimento dei dati sono le uniche operazioni eseguite sui database.

Per la versione Stand Alone lo script è il seguente:

```
CREATE SCHEMA IF NOT EXISTS `setteemezzo_standalone`;  
  
CREATE TABLE IF NOT EXISTS `setteemezzo_standalone`.`transazione` (  
  `id_partita` INT(11) NOT NULL AUTO_INCREMENT,  
  `giocatore` VARCHAR(20) NOT NULL,  
  `ruolo` VARCHAR(8) NOT NULL DEFAULT '',  
  `punteggio` DOUBLE DEFAULT NULL,  
  `vincita` VARCHAR(6) NOT NULL DEFAULT '0',  
  `data` DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',  
  PRIMARY KEY (`id_partita`)  
)  
ENGINE = InnoDB;  
  
GRANT SELECT, INSERT  
  ON `setteemezzo_standalone`.*  
  TO 'user'@'localhost'  
  IDENTIFIED BY 'user';
```

Per la versione Client/Server lo script è il seguente

```
CREATE SCHEMA IF NOT EXISTS `setteemezzo`;  
  
CREATE TABLE IF NOT EXISTS `setteemezzo`.`utente` (  
  `email` VARCHAR(20) NOT NULL,  
  `nickname` VARCHAR(20) NOT NULL,  
  `password` VARCHAR(10) NOT NULL,  
  PRIMARY KEY (`email`)  
)  
ENGINE = InnoDB;  
  
CREATE TABLE IF NOT EXISTS `setteemezzo`.`transazione` (  
  `id_partita` INT(11) NOT NULL AUTO_INCREMENT,  
  `giocatore` VARCHAR(20) NOT NULL,  
  `ruolo` VARCHAR(8) NOT NULL DEFAULT '',  
  `punteggio` DOUBLE DEFAULT NULL,  
  `vincita` VARCHAR(6) NOT NULL DEFAULT '0',  
  `data` DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',  
  FOREIGN KEY (`giocatore`) REFERENCES `utente` (`email`)  
  ON DELETE CASCADE,  
  PRIMARY KEY (`id_partita`)  
)  
ENGINE = InnoDB;  
  
GRANT SELECT, INSERT  
  ON `setteemezzo`.*  
  TO 'user'@'localhost'  
IDENTIFIED BY 'user';
```



## 6.5. Pagine JSP

Nell'estensione da noi sviluppata, è possibile solo per gli utenti registrati al gioco usufruire delle funzioni di salvataggio, caricamento e visualizzazione storico delle partite. Tale estensione è presente solo nel prodotto Client/Server, in quanto non risulta necessario nel prodotto Stand Alone. Per consentire all'utente di registrarsi, è stata progettata una pagina [JSP](#) ed un form di registrazione. Un'ulteriore pagina JSP è stata usata per consentire la visione dello storico di un utente.

### 6.5.1. SaveUser.jsp

La pagina JSP in esame si occupa della registrazione di un nuovo utente e della visualizzazione del riepilogo dei dati nel caso la registrazione vada a buon fine. Inizialmente viene presentata una pagina html che visualizza un form di registrazione. Nel form sono presenti due pulsanti: uno consente di resettare tutti i campi del form riempiti, l'altro serve a inviare la richiesta di registrazione. In seguito alla pressione di quest'ultimo pulsante, viene richiamata la pagina *SaveUser.jsp*, passandogli come parametri i dati forniti dall'utente. I dati vengono successivamente salvati in un oggetto della classe *UtenteEntry* grazie alle seguenti istruzioni:

```
<jsp:useBean id="user" scope="page" class="database.UtenteEntry">
    <jsp:setProperty name="user" property="*" />
</jsp:useBean>
```

Con il tag `<jsp:useBean>` indichiamo di voler salvare i dati contenuti nella richiesta in un oggetto della classe *UtenteEntry* denominato *user*. Con il tag `<jsp:setProperty>` indichiamo che deve salvare tutti i campi che costituiscono l'oggetto.

Successivamente viene verificata l'esistenza di un utente con la stessa password e la stessa email inserita nel form: se già presente viene presentato un avviso ed inclusa la pagina html contenente il form, altrimenti viene salvato l'utente e presentato una pagina di riepilogo. Se non è possibile portare a compimento la registrazione, viene presentato un'ulteriore avviso.

### 6.5.2. Storico.jsp

*Storico.jsp* si occupa invece della presentazione dello storico delle partite giocate da uno specifico giocatore. Per eseguire tale compito, inizialmente viene ricavata la data in cui si vuole visionare lo storico dalla richiesta http. La richiesta viene creata in seguito alla scelta della data dal calendario presentato con la pressione della voce “Storico” presente nella barra dei menù dell’applet. Dopo il recupero della data viene eseguito un algoritmo che consente all’utente di visionare non solo lo storico della data scelta, ma anche delle date precedenti o successive a quella selezionata.

Se la pagina viene visitata per la prima volta, allora viene ricavata dalla richiesta l’email dell’utente, salvata assieme alla data selezionata nella sessione, recuperate le partite e presentate sotto forma tabulare.

Se la pagina viene rivisitata vuol dire che l’utente ha selezionato una delle due frecce, poste sulla tabella di riepilogo delle partite, che consente di navigare lo storico. Nel caso l’utente voglia visitare la data successiva a quella scelta, la richiesta presentata alla pagina sarà del tipo “*Storico.jsp?data=succ*”; l’algoritmo allora recupera dalla sessione i dati salvati, incrementa la data di un giorno, salva la data modificata e aggiorna lo storico. Nel caso l’utente voglia la data precedente a quella selezionata, la richiesta sarà del tipo “*Storico.jsp?data=prec*”; le operazioni compiute sono le stesse presentate per il caso di data successiva, solo che la data viene decrementata di un giorno e non incrementata.

## 7. Guide

Presentiamo di seguito le guide all'uso di entrambi i prodotti.

### 7.1. Svolgimento del gioco

All'inizio della mano il mazziere distribuisce una carta coperta allo sfidante. Lo sfidante effettua le seguenti operazioni:

1. Guarda la propria carta.
2. Effettua la puntata ponendo la somma corrispondente sopra la carta coperta o davanti a sé. Se si sballa si danno i soldi e le carte in mano.
3. Se lo vuole, può richiedere altre carte per migliorare il proprio punteggio. Tutte le carte successive alla prima vengono date scoperte una per volta, fino a quando lo sfidante continua a richiederne.
4. Se lo sfidante sballa o realizza 7 e mezzo deve farlo notare immediatamente scoprendo anche la prima carta ricevuta. Quando si sballa, la posta viene subito ritirata dal banco, in caso contrario il gioco procede e il turno passa al banco.

Il mazziere scopre la propria carta e decide se prenderne altre. Se il mazziere sballa, paga alla pari la puntata dello sfidante se ancora in gioco. In caso contrario, lo sfidante scopre la propria carta e confronta il punteggio con quello del mazziere. Lo sfidante cede, quindi, la propria posta al banco o ne incassa l'equivalente secondo le regole del gioco. Con il sette e mezzo reale la posta viene raddoppiata. Se il sette e mezzo viene pagato doppio, dà diritto di diventare mazziere allo sfidante; in caso contrario il mazziere tiene il banco fino all'esaurimento delle carte nel mazzo, dopodiché il banco passa al giocatore seduto alla sua destra.

## 7.2. Guida versione Stand Alone

### 7.2.1. Installazione su sistema operativo Windows

Tutti gli applicativi necessari all'avvio del gioco sono presenti nella cartella "win".

1. Procedere all'installazione della JRE 6 (Java Runtime Environment versione 6) avviando il file "jre-6u14-windows-i586.exe" e seguendo le istruzioni riportate sullo schermo. Se si è in possesso del programma, controllare la versione e aggiornare il software nel caso si disponga di una versione inferiore alla 6. Se diversamente si dispone di una JRE 6, passare al punto 2.
2. Procedere all'installazione del dbms MySQL versione 5.0 avviando il file "mysql\_server\_5.0.exe" e seguendo le istruzioni riportate sullo schermo. Se si è in possesso del programma, controllare la versione e aggiornare il software nel caso si disponga di una versione inferiore alla 5.0. Se diversamente si dispone di MySQL 5.0, passare al punto 3.
3. Procedere alla creazione del database necessario al salvataggio dello storico delle partite effettuate. La procedura di caricamento è stata automatizzata: basta avviare il file "Installer.bat" e inserire quando richiesta la password impostata durante l'installazione di MySQL. Se invece, si è in grado di utilizzare il dbms MySQL, caricare lo script "setteemezzo\_standalone.sql".

### 7.2.2. Installazione su sistema operativo Linux

Tutti gli applicativi necessari all'avvio del gioco sono presenti nella cartella "linux".

1. Procedere all'installazione della JRE 6 (Java Runtime Environment versione 6) attraverso il gestore di pacchetti fornito dal sistema operativo e seguendo le istruzioni riportate sullo schermo. Se si è in possesso del programma, controllare la versione e aggiornare il software nel caso si disponga di una versione inferiore alla 6. Se diversamente si dispone di una JRE 6, passare al punto 2.
2. Procedere all'installazione del dbms MySQL versione 5.0 attraverso il gestore di pacchetti fornito dal sistema operativo e seguendo le istruzioni riportate sullo schermo. Se si è in possesso del programma, controllare la versione e aggiornare il software nel caso si disponga di una versione inferiore alla 5.0. Se diversamente si dispone di MySQL 5.0, passare al punto 3.
3. Procedere alla creazione del database necessario al salvataggio dello storico delle partite effettuate. La procedura di caricamento è stata automatizzata: eseguire il file "Installer.sh" tramite il comando "sh Installer.sh". Digitare, quando richiesta, la password impostata durante l'installazione di MySQL. Se invece, si è in grado di utilizzare il dbms MySQL, caricare lo script "setteemezzo\_standalone.sql".

### 7.2.3. Avviare Sette e Mezzo

Puoi avviare il gioco nei seguenti modi, a seconda del sistema operativo:

#### Sistema Windows

Accedere alla cartella "win". Doppio click sul file

```
avvia.bat
```

#### Sistema Linux

Accedere alla cartella "linux".

##### **1° Metodo**

Aprire un terminale e rendere eseguibile lo script di avvio con il comando

```
chmod +x avvia.sh
```

Sempre nel terminale eseguire lo script tramite il comando

```
./avvia.sh
```

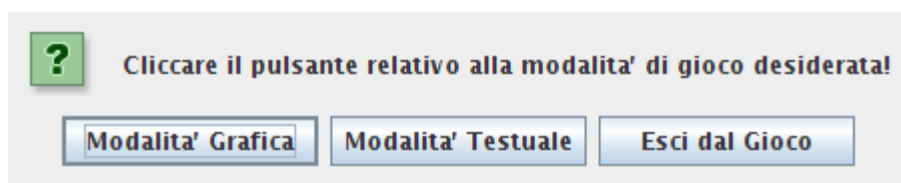
##### **2° Metodo**

Avviare lo script da terminale tramite il comando

```
sh avvia.sh
```

#### 7.2.4. Quando si avvia Sette e Mezzo

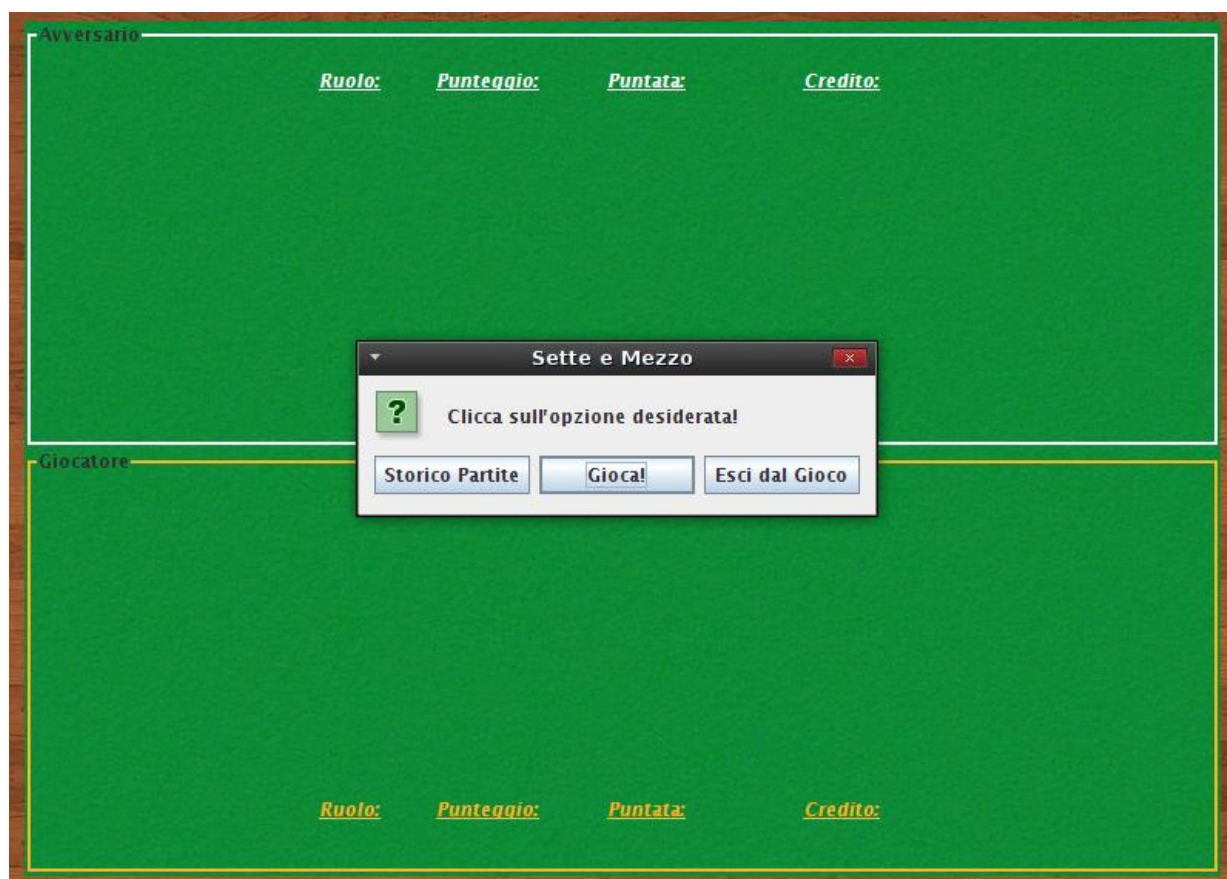
Quando si avvia Sette e Mezzo, viene mostrata la seguente finestra.



Per avviare il gioco in modalità grafica premere il pulsante “Modalità Grafica”. Se si desidera avviare il gioco in modalità testuale, ossia giocabile tramite terminale, premere il pulsante “Modalità Testuale”. Nel caso si volesse uscire dal gioco, premere il pulsante “Esci dal Gioco”.

#### Modalità grafica

Selezionando la modalità grafica, viene mostrata la seguente finestra.



La finestra del gioco si presenta costituita da due elementi: il tavolo dell'avversario e il tavolo del giocatore. Ognuno di questi due tavoli si presenta, a sua volta, suddiviso in due aree: area informazioni e area carte sul tavolo.

Nell'area informazioni vengono visualizzate le seguenti informazioni: il ruolo del giocatore, il punteggio della mano corrente, la puntata effettuata e il credito disponibile. Per quanto riguarda il punteggio, nel tavolo del giocatore sarà pari al punteggio totale delle carte in suo possesso; per

l'avversario, invece, sarà pari al punteggio delle sole carte scoperte. Nell'area carte sul tavolo verranno visualizzate le carte ricevute durante una mano.

Ogni tavolo presenta anche il nome del giocatore associato. Nel caso dell'avversario il nome sarà CPU indipendentemente dal ruolo impersonato dall'avversario gestito dal computer.

Viene anche visualizzata una finestra di scelta con tre opzioni: "Storico Partite", "Gioca!" ed "Esci dal Gioco". Selezionando "Gioca!" si inizierà a giocare, mentre selezionando "Esci dal Gioco" si chiuderà l'applicazione. Con "Storico Partite" si potranno visualizzare tutte le informazioni del giocatore relative alle partite giocate in una determinata data. Alla selezione dell'opzione, viene



visualizzato un calendario.

Selezionando una data, verrà mostrata una tabella con le informazioni sulle partite giocate.

Avversario				
Storico Partite				
Nome giocatore	Ruolo	Punteggio	Vincita	Data
cry	Sfidante	7.5	-12	27-05-2009 - 11:42:14
cry	Sfidante	2.0	-12	27-05-2009 - 15:28:38
cry	Sfidante	4.0	-12	27-05-2009 - 15:28:38
cry	Sfidante	7.0	+12	27-05-2009 - 15:34:02
cry	Sfidante	10.0	-12	27-05-2009 - 15:34:02
cry	Sfidante	7.0	+12	27-05-2009 - 15:41:47
cry	Sfidante	7.0	-12	27-05-2009 - 15:41:47
Cry	Sfidante	8.0	-0	27-05-2009 - 16:29:16
cry	Sfidante	8.5	-12	27-05-2009 - 16:34:53
da	Sfidante	3.0	-12	27-05-2009 - 17:47:52
a	Sfidante	6.0	-1	27-05-2009 - 18:16:33

### Modalità testuale

Selezionando la modalità testuale, viene mostrato nel terminale la seguente videata.

```
File Modifica Visualizza Terminale Schede Ajuto

=====
ZAMEZZO
=====
Autori in ordine alfabetico: Tarantino - Turturo
=====

Digita:
0. se desideri visionare lo storico
1. se desideri giocare
2. se desideri uscire dal gioco
```



Digitando il numero 0, invece, si potrà interagire con lo storico delle partite giocate dal giocatore. Alla selezione di questa opzione, viene richiesta la data di cui si desidera visionare lo storico nel formato europeo. Dopo averla inserita verrà visualizzato lo storico.

```
File Modifica Visualizza Terminale Schede Ajuto
//_/_|||_|_/_/_/_/_|_|_|
//\_/_||_|_|_/_/_/_\_|_|_|

=====
Autori in ordine alfabetico: Tarantino - Turturo
=====

Digita:
0. se desideri visionare lo storico
1. se desideri giocare
2. se desiderare uscire dal gioco

0

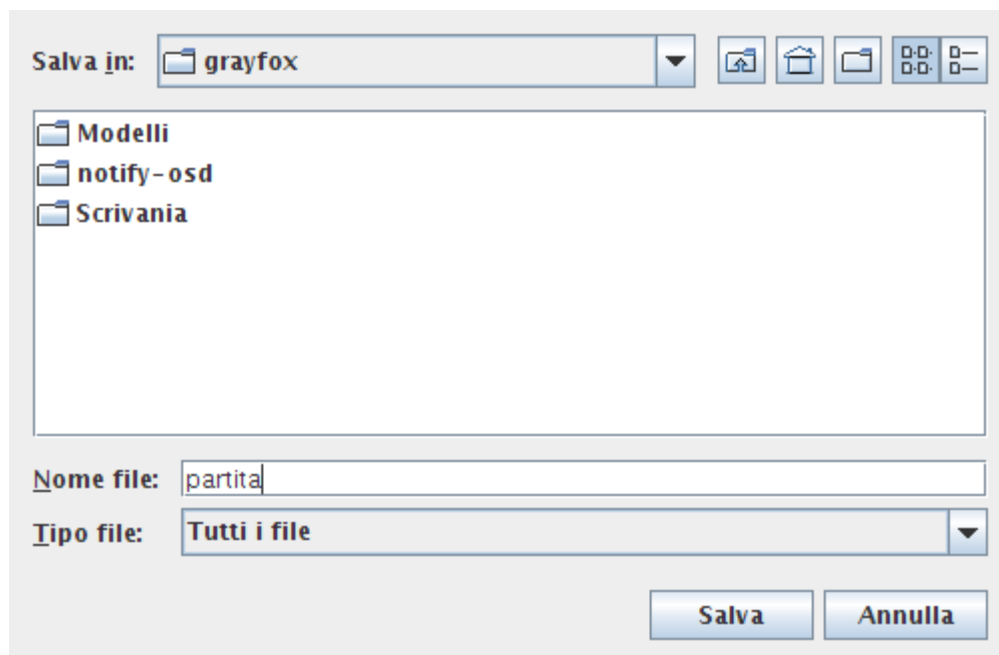
Digita la data per il quale desideri
visionare lo storico delle partite nel formato GG-MM-AAAA
27-05-2009

[
Giocatore: cry ruolo: Sfidante punteggio: 7.5 vincita: -12 data: 2009-05-27
11:42:14.0,
Giocatore: cry ruolo: Sfidante punteggio: 2.0 vincita: -12 data: 2009-05-27
15:28:38.0,
Giocatore: cry ruolo: Sfidante punteggio: 4.0 vincita: -12 data: 2009-05-27
15:28:38.0,
Giocatore: cry ruolo: Sfidante punteggio: 7.0 vincita: +12 data: 2009-05-27
15:34:02.0,
Giocatore: cry ruolo: Sfidante punteggio: 10.0 vincita: -12 data: 2009-05-27
15:34:02.0,
Giocatore: cry ruolo: Sfidante punteggio: 7.0 vincita: +12 data: 2009-05-27
15:41:47.0,
Giocatore: cry ruolo: Sfidante punteggio: 7.0 vincita: -12 data: 2009-05-27
15:41:47.0,
Giocatore: Cry ruolo: Sfidante punteggio: 8.0 vincita: -0 data: 2009-05-27
16:29:16.0,
Giocatore: cry ruolo: Sfidante punteggio: 8.5 vincita: -12 data: 2009-05-27
16:34:53.0,
Giocatore: da ruolo: Sfidante punteggio: 3.0 vincita: -12 data: 2009-05-27
17:47:52.0,
Giocatore: a ruolo: Sfidante punteggio: 6.0 vincita: -1 data: 2009-05-27 18
:16:33.0]
```

### 7.2.5. Salvare una partita

Ogni qual volta si vuole uscire dall'applicazione dopo aver effettuato una partita, verrà chiesto di salvare una partita. Se si risponde in modo affermativo, verrà salvato sul computer un file con estensione ".dat" contenente le informazioni sulla partita.

In modalità grafica comporterà l'apertura di una finestra in cui specificare la posizione e il nome del file di salvataggio.

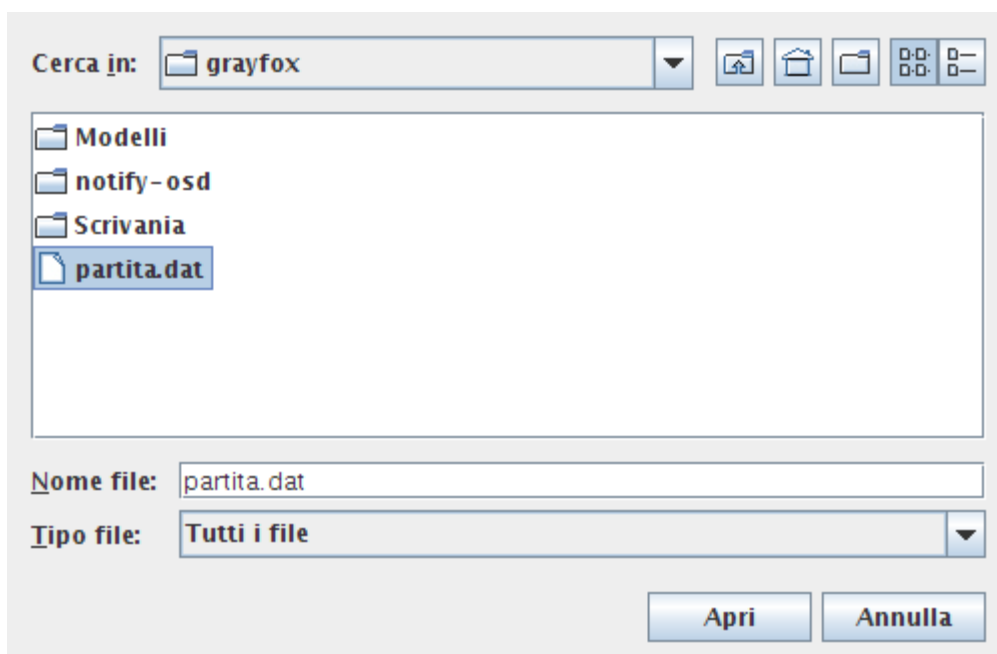


Nella modalità testuale invece verrà chiesto solo il nome del file. Il file verrà automaticamente salvato nella cartella di avvio del programma.

### 7.2.6. Caricare una partita

Il caricamento della partita viene chiesto ogni qual volta si vuole iniziare a giocare.

In modalità grafica verrà aperta una finestra in cui selezionare il file ".dat" in cui è stata salvata la partita.



Nella modalità testuale, invece, viene presentata una lista dei file con estensione ".dat" presenti nella stessa cartella di avvio del programma; basterà digitare uno dei nomi dei file (completo di estensione) per poter continuare la partita.

### 7.3. Guida versione Client/Server lato client

#### 7.3.1. Avviare Sette e Mezzo

Per avviare il gioco basterà semplicemente fare doppio clic sul file "SetteEMezzo.html". Questa operazione porterà all'apertura del proprio browser web predefinito e alla visualizzazione della schermata principale del gioco.



La finestra di gioco del Sette e Mezzo contiene i seguenti componenti:

##### **Barra dei menù**

La barra dei menù presenta tutti i comandi necessari a giocare a SetteEMezzo.

##### **Area di gioco**

L'area di gioco permette la visualizzazione dello stato dei giocatori in una mano corrente e in una partita.

### 7.3.2. Barra dei menù

La barra dei menù, posizionata sopra l'area di gioco, presenta i seguenti menù:

#### Partita

Questo menù contiene le seguenti voci:

- **Nuova partita (Ctrl-N):** pulisce il tavolo e inizia una nuova partita
- **Carica partita (Ctrl-O):** carica una partita precedentemente salvata. Questa funzione è abilitata per i soli registrati
- **Salva partita (Ctrl-S):** salva la partita che si sta giocando. Questa funzione è abilitata per i soli registrati
- **Login (Ctrl-L):** consente l'autenticazione dell'utente
- **Registrazione (Ctrl-R):** consente di registrarsi per poter usufruire delle funzionalità di salvataggio, caricamento e visualizzazione storico.
- **Storico (Ctrl-T):** consente di visualizzare un resoconto di tutte le mani delle partite effettuate nel gioco desiderato.

#### Aspetto

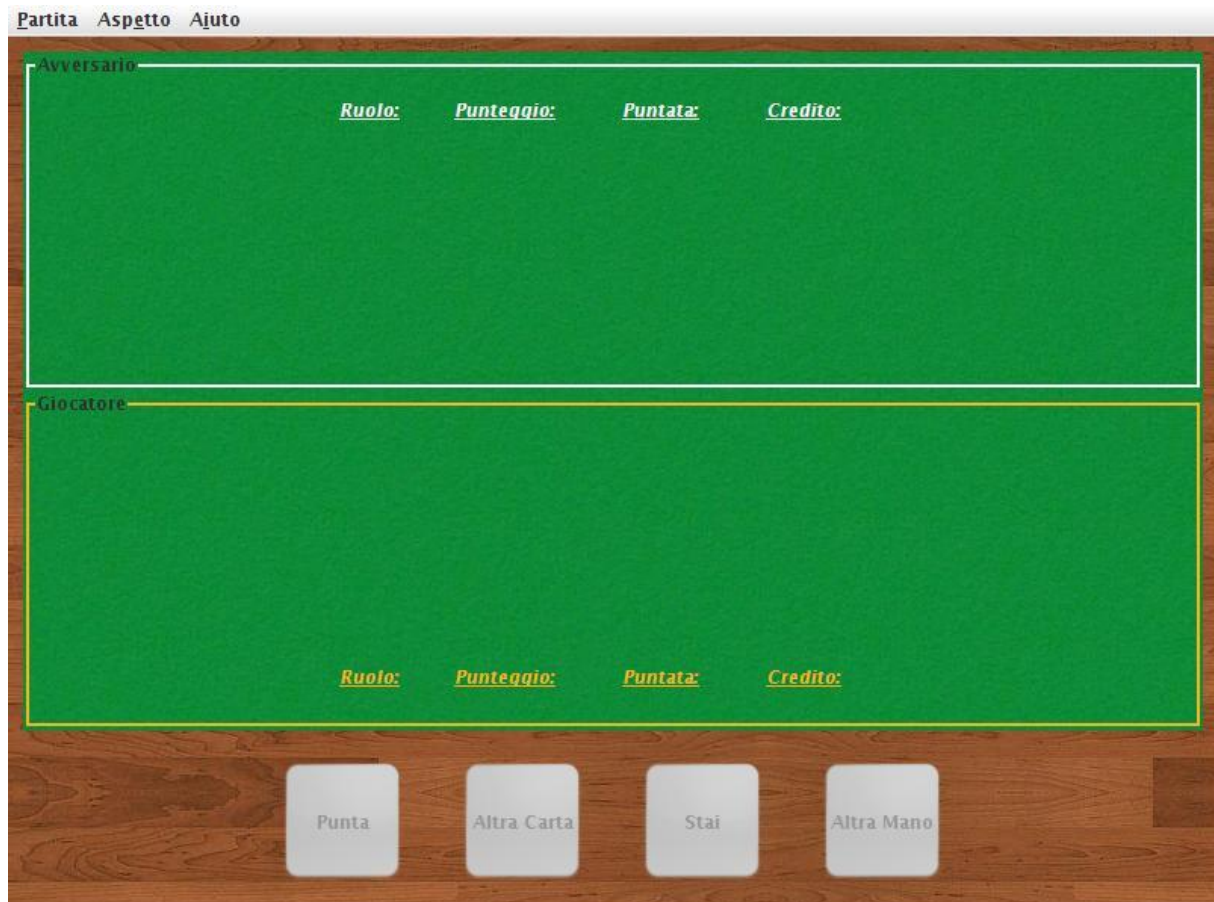
Questo menù consente di cambiare il look and feel dell'applicazione, ossia lo stile della barra dei menù e delle finestre di dialogo.

#### Aiuto

Questo menù contiene le seguenti voci:

- **Guida (F1):** apre il tuo help browser e mostra questa guida.
- **Informazioni:** apre l'about dialog, mostrando il nome degli autori accompagnato dal logo del gioco.

### 7.3.3. Area di gioco



L'area di gioco si presenta composta dal tavolo dell'avversario e del giocatore. Ognuno di questi due tavoli si presenta, a sua volta, suddiviso in due aree: **area informazioni** e **area carte sul tavolo**. Nell'area informazioni viene visualizzato il ruolo del giocatore, il punteggio della mano corrente, la puntata effettuata e il credito disponibile. Per quanto riguarda il punteggio, nel tavolo del giocatore sarà pari al punteggio totale delle carte in suo possesso; per l'avversario, invece, sarà pari al punteggio delle sole carte scoperte. Nell'area carte sul tavolo verranno visualizzate le carte ricevute durante una mano. Ogni tavolo presenta anche il nome del giocatore associato. Nel caso dell'avversario il nome sarà **CPU** indipendentemente dal ruolo impersonato. L'area di gioco si presenterà così strutturata solo dopo aver iniziato una partita.

#### Pulsanti di interazione

Permettono all'utente di interagire con il gioco. Sono quattro:

- **Punta:** consente al giocatore di effettuare la sua puntata.
- **Altra Carta:** consente al giocatore di richiedere un'altra carta.
- **Stai:** permette al giocatore di terminare il turno di gioco.
- **Altra Mano:** consente al giocatore di giocare una nuova mano.

I pulsanti saranno visibili solo dopo aver iniziato una partita. I pulsanti saranno abilitati e disabilitati a seconda del contesto di gioco

#### 7.3.4. Come registrarsi

La registrazione permette all'utente di usufruire delle funzionalità avanzate fornite dal sistema:

- scelta di un nickname fisso
- salvataggio delle partite
- caricamento delle partite
- consultazione dello storico delle partite giocate

La registrazione al sistema risulta semplice. Selezionare dalla barra dei menù la voce Partita; dal menù a tendina apparso selezionare la voce "Registrazione". Questa operazione porterà alla visualizzazione di una nuova pagina web contenente un form di registrazione.

La compilazione del form di registrazione è guidata attraverso controlli previsti per ciascun campo.

È possibile selezionare anche la lingua inglese. ATTENZIONE, tale selezione non modificherà la lingua del gioco che, per questa versione, rimarrà sempre in italiano

**Modulo di registrazione**

Lingua: Italiano ▼  
Italiano  
Inglese

---

Nickname:

Password:

Conferma Password:

Email:

Conferma Email:

Compilato il form con le informazioni necessarie, sarà presentata una pagina di riepilogo con tutte le informazioni inserite.

# Registrazione avvenuta con successo!

## Riepilogo dati

**Nickname:** utente

**Password:** utente

**Email:** utente@dominio.ext

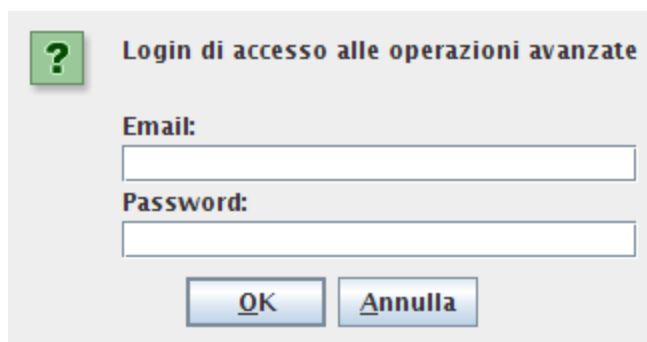


### 7.3.5. Come autenticarsi

L'utente potrà autenticarsi al sistema in due modi:

1. Scegliendo la voce "Login" nel menù "Partita"
2. Selezionando una delle tre opzioni previste per gli utenti registrati ("Salva partita", "Carica partita" e "Storico")

Entrambe le soluzioni portano alla visualizzazione di una finestra di dialogo in cui inserire i propri dati di accesso (email e password definiti in fase di registrazione).

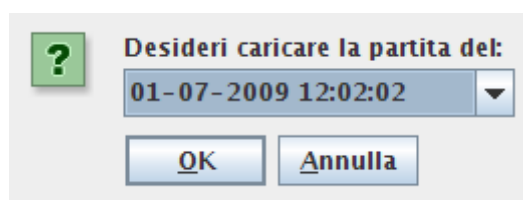


### 7.3.6. Salvare una partita

Ogni qual volta lo si ritenga necessario, l'utente registrato potrà salvare la partita. Un messaggio confermerà l'avvenuto salvataggio o notificherà l'impossibilità di tale operazione. Per salvare la partita selezionare "Salva partita" dal menù "Partita". Il sistema consentirà il salvataggio di sole dieci partite; terminati gli slot liberi a disposizione, il sistema cancellerà in maniera automatica i salvataggi più vecchi.

### 7.3.7. Caricare una partita

Per caricare una partita selezionare "Carica partita" dal menù "Partita". Selezionando tale opzione verrà mostrata una finestra di dialogo in cui selezionare la partita da caricare.



Selezionando il menù a tendina saranno presentate le ultime dieci partite salvate ordinate per data di salvataggio. Basterà selezionarne una e cliccare sul pulsante "OK" per caricare la partita desiderata.

### 7.3.8. Storico personale delle partite

L'utente registrato potrà visualizzare lo storico delle partite giocate. Per accedere a tale funzionalità, selezionare dal menù a tendina "Partita" la voce "Storico". Apparirà una finestra di dialogo nella quale l'utente potrà selezionare la data di cui desidera visualizzare lo storico.

Clicca due volte sul giorno per il quale desideri visionare lo storico

luglio

2009

lun	mar	mer	gio	ven	sab	dom
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Scelta la data si aprirà una nuova finestra del browser web di default che permetterà la visualizzazione delle partite giocate sotto forma tabellare. Le righe che nella tabella compaiono dello stesso colore, denotano lo storico delle mani di una stessa partita.

L'utente avrà anche la possibilità di scorrere lo storico delle partite sostenute semplicemente interagendo con le frecce posizionate in alto nella tabella.

Precedente

Data Storico: 02/07/2009

Successivo

Ruolo	Punteggio	Vincita	Data
Sfidante	7.0	-1 Euro	02/07/2009 11:17:18
Sfidante	9.5	-1 Euro	02/07/2009 11:17:18
Sfidante	7.0	-20053 Euro	02/07/2009 11:19:21
Sfidante	9.0	-2755 Euro	02/07/2009 11:19:21
Banco	7.0	+1 Euro	02/07/2009 11:19:21
Sfidante	6.5	-1 Euro	02/07/2009 11:27:16
Sfidante	7.5	+1 Euro	02/07/2009 11:27:16
Banco	5.0	+6400 Euro	02/07/2009 11:27:16
Banco	3.0	-5760 Euro	02/07/2009 11:27:16
Sfidante	5.0	-1 Euro	02/07/2009 11:40:16
Sfidante	6.0	-1 Euro	02/07/2009 11:40:16
Sfidante	11.0	-1 Euro	02/07/2009 11:40:16
Sfidante	8.5	-1 Euro	02/07/2009 11:40:16
Sfidante	9.0	-1 Euro	02/07/2009 11:40:16
Banco	7.0	+1 Euro	02/07/2009 11:40:16

nomeutente@dominio.ext

#### 7.4. Guida all'installazione dell'applicazione Client/Server lato Server

N.B. TUTTI I FILE BAT SOTTO SPECIFICATI DOVRANNO ESSERE ESEGUITI APRENDO CON DIRITTI AMMINISTRATIVI NELLA CARTELLA DI INSTALLAZIONE "SetteEMezzo - Client\_Server/win/"

1. Installare MySQL Server versione 5.0 o superiore;
2. Installare Java JRE versione 1.6 o superiore;
3. Installare Tomcat versione 6.0.20 avviando il file eseguibile "apache-tomcat-6.0.20.exe";
4. A questo punto, se tutto è andato per il verso giusto, puntando all'indirizzo <http://localhost:8080> si dovrebbe avere la pagina di benvenuto di Tomcat;
5. Eseguire lo script Installer.bat, eseguendo un doppio click sul file ed eseguire le istruzioni mostrate.
6. Modificare il file di configurazione "tomcat-users.xml", che si trova nella cartella "conf" presente nella directory di installazione del web server Tomcat, e aggiungere il proprio ruolo.

Per esempio:

```
<role rolename="manager"/>
```

```
<user username="nomeUtente" password="passwordUtente" roles="manager"/>
```

7. Effettuare il deploy del file JSPMAP.war accedendo alla voce Tomcat Manager dalla pagina di benvenuto precedentemente menzionata.

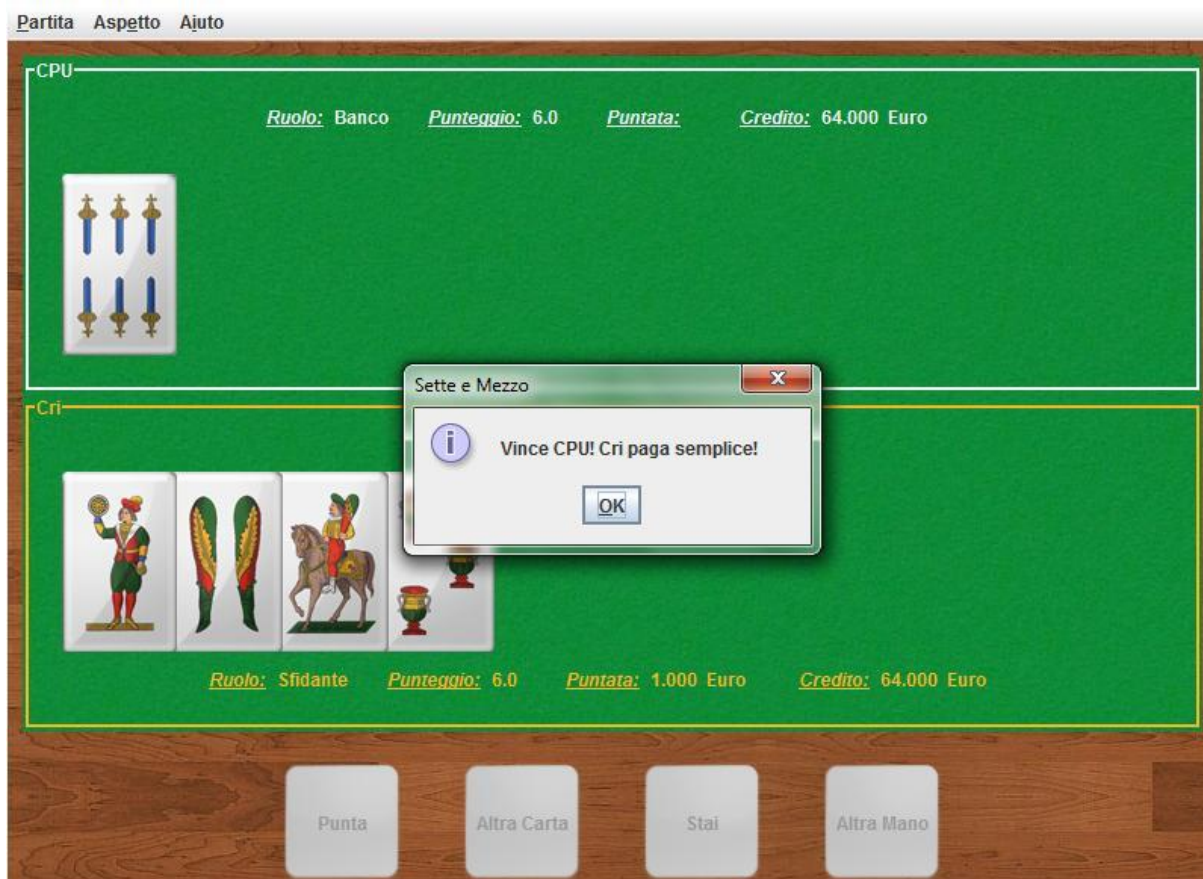
Per avviare l'applicazione lato server di Sette e Mezzo, eseguire lo script `avvioServer.bat`.

## 8. Esempi di test

### 8.1. Entrambi i giocatori realizzano lo stesso punteggio: vince il banco

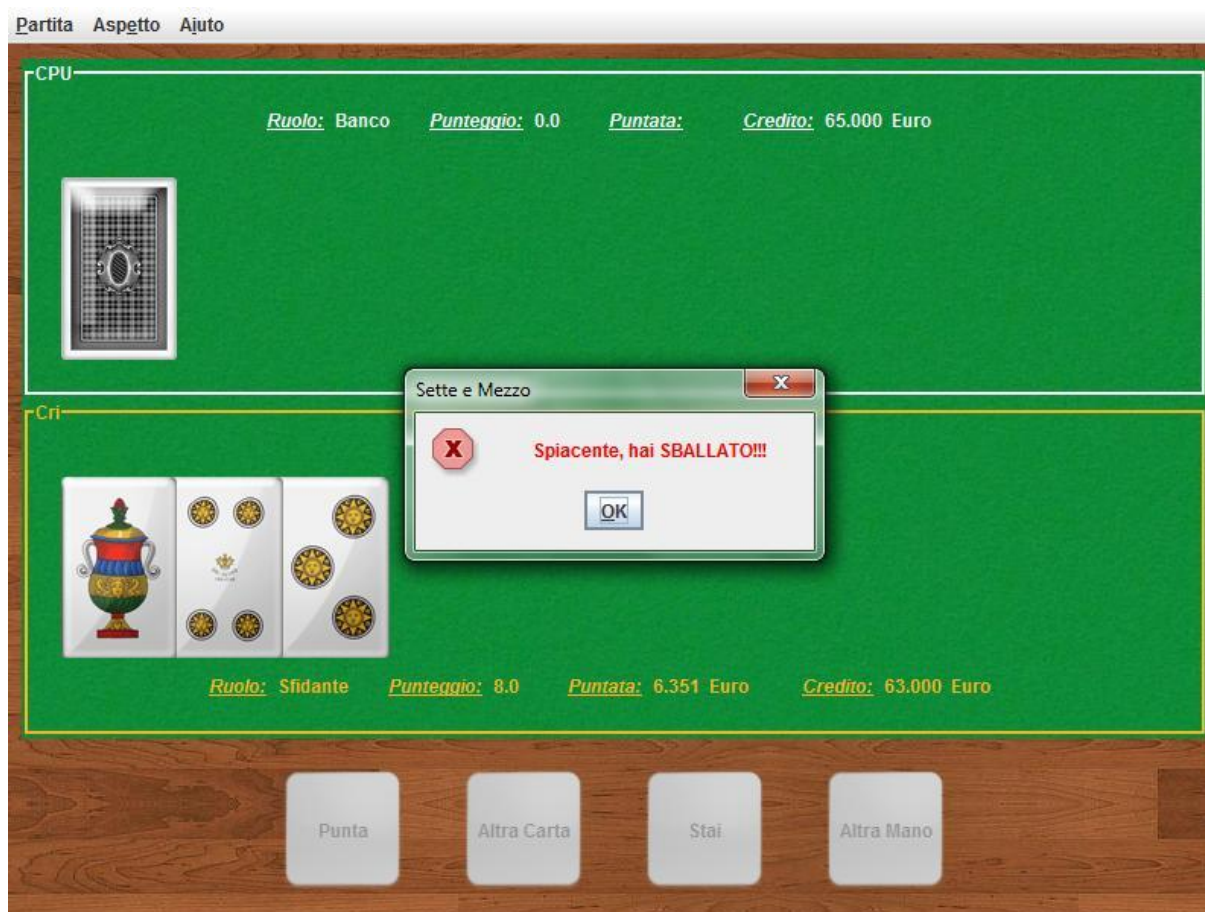
La matta non è stata estratta. Il punteggio del giocatore, nel ruolo di sfidante, e quello della CPU, nel ruolo del banco, è identico.

In questa situazione è il giocatore con ruolo di banco a vincere una posta semplice.



## 8.2. Lo sfidante sballa: vince il banco

La matta non è stata estratta. Il punteggio del giocatore, nel ruolo di sfidante, è maggiore di 7  $\frac{1}{2}$ . Lo sfidante ha, dunque, sballato. Il banco CPU, di conseguenza, non gioca.



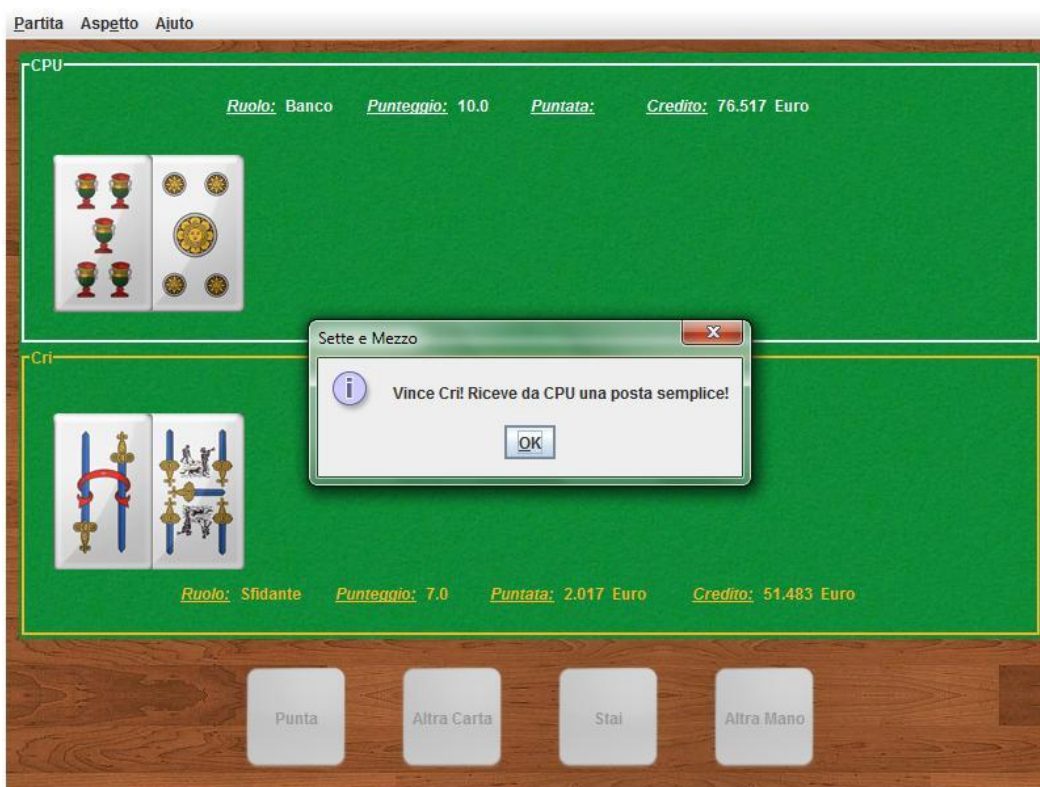


### 8.3. Il banco sballa: vince lo sfidante

La CPU, nel ruolo di banco, sballa.



Il giocatore, nel ruolo di sfidante, riceve dal banco una posta semplice.



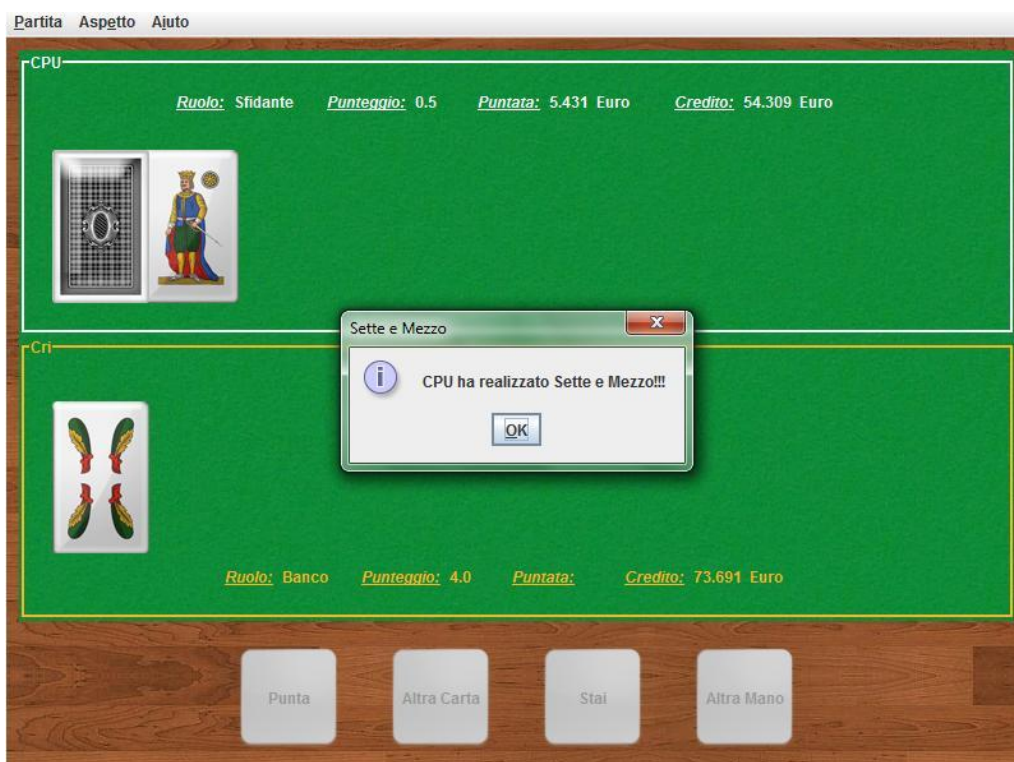
#### 8.4. Lo sfidante totalizza un punteggio superiore al banco: vince lo sfidante

La CPU, nel ruolo di sfidante, realizza un punteggio inferiore a quello del giocatore nel ruolo di banco.

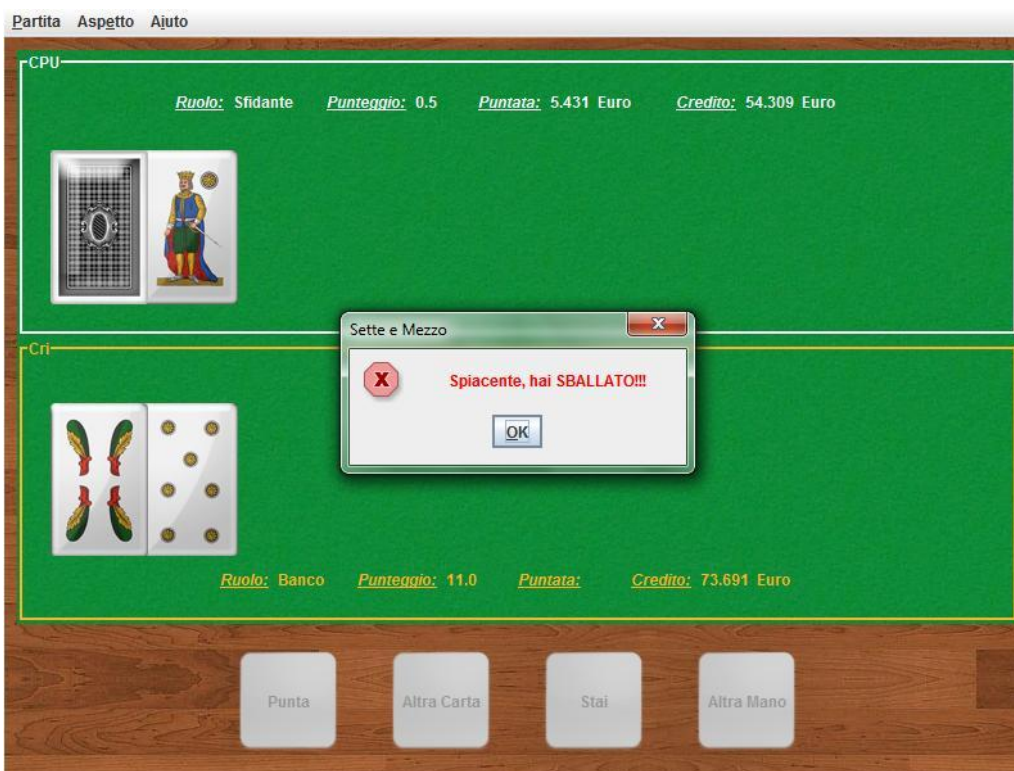


**8.5. Lo sfidante totalizza 7 ½ con matta e figura: vince lo sfidante che diventa banco. Il mazzo viene mescolato**

La CPU, nel ruolo di sfidante, realizza 7 ½ con matta e figura.



Il giocatore, nel ruolo di banco, invece sballa.





Quindi vince lo sfidante CPU, ricevendo una posta doppia dal banco.



Il banco diventa sfidante alla mano successiva (per le regole del gioco).



Poiché è stata estratta la matta, come previsto dalle regole del gioco, il mazzo viene mescolato. Inoltre è possibile notare come sia avvenuto lo scambio di ruoli.



### 8.6. Lo sfidante totalizza 7 ½ e il banco sballa: vince lo sfidante

Lo sfidante realizza il punteggio di 7 ½ con più di due carte mentre il banco realizza un punteggio maggiore di 7 ½. Il vincitore della mano è lo sfidante.





**8.7. Lo sfidante totalizza 7 ½ e il banco totalizza un punteggio inferiore a 7 ½: vince lo sfidante e diventa banco**

Lo sfidante realizza il punteggio di 7 ½ con matta e figura mentre il banco realizza un punteggio inferiore di 7 ½. Il vincitore della mano è lo sfidante il quale diventa banco nella mano successiva.



## 8.8. Errori di Compilazione Form

1. L'utente omette campi obbligatori;

**Modulo di registrazione**

Lingua Italiano ▾

---

*Il campo Password e' obbligatorio*

Nickname:  ✖

Password:  ✖

Conferma Password:

Email:

Conferma Email:

2. L'utente inserisce una password minore di 6 o maggiore di 10 caratteri;

**Modulo di registrazione**

Lingua Italiano ▾

---

*Attenzione! La password deve avere tra 6 e 10 caratteri*

Nickname:  ✔

Password:  ✖

Conferma Password:

Email:

Conferma Email:

3. L'utente inserisce un formato d'indirizzo e-mail non valido;

**Modulo di registrazione**

Lingua Italiano ▾

---

*Inserire un indirizzo email valido*

**Nickname:**  ✓

**Password:**  ✓

**Conferma Password:**  ✓

**Email:**  ✗

**Conferma Email:**  ✗

4. L'utente inserisce un'e-mail di conferma diversa da quella precedentemente indicata;

**Modulo di registrazione**

Lingua Italiano ▾

---

*L'indirizzo email confermato e' diverso da quello scelto, controllare*

**Nickname:**  ✓

**Password:**  ✓

**Conferma Password:**  ✓

**Email:**  ✓

**Conferma Email:**  ✗

### 8.9. Indirizzo e-mail già presente nel database del sistema

L'utente tenta di registrarsi al sistema con un indirizzo e-mail già utilizzato in un'altra registrazione.

*Modulo di registrazione*

Lingua Italiano ▾

---

Nickname:  ✓

Password:  ✓

Conferma Password:  ✓

Email:  ✓

Conferma Email:  ✓

**Attenzione! Esiste già un account associato a user@utente.it**

*Modulo di registrazione*

Lingua Italiano ▾

---

Nickname:

Password:

Conferma Password:

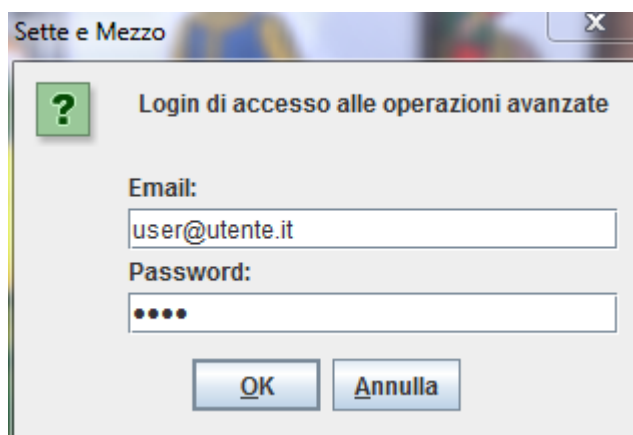
Email:

Conferma Email:

## 8.10. Errori di Accesso al Sistema

Questo problema si verifica quando:

- L'utente non registrato tenta di accedere al sistema;
- L'utente commette errori di digitazione delle proprie credenziali di accesso ai servizi aggiuntivi.





## 9. Sviluppi futuri

Nelle prossime versioni del gioco sono previsti i seguenti sviluppi:

- Riprogettazione dell'interazione, in particolare nella versione Stand Alone.
- Implementazione di un help in linea per la versione Stand Alone.
- Implementazione di un sistema di aggiornamento integrato per la versione Stand Alone.
- Implementazione di un sistema di analisi statistica dei risultati delle partite in sostituzione dello storico delle stesse.
- Implementazione di un mazzo di carte francesi come mazzo alternativo nel gioco del Sette e Mezzo.
- Possibilità di scegliere tra diversi tipi di mazzi di carte italiani con cui giocare (es. Sarde, Piemontesi, ecc.).
- Supporto alle lingue straniere.
- Miglioramento del sistema di salvataggio delle partite, permettendo all'utente la cancellazione delle partite e la loro sovrascrittura.
- Implementazione di un sistema di cifratura per la password degli utenti registrati.
- Implementazione del gioco in versione multi – utente, permettendo ad un utente registrato di giocare contro altri utenti registrati.
- Implementazione di un sistema di chat interna al gioco (legata alla implementazione della versione multi–utente).

## 10. Glossario

Riportiamo qui i termini che sono stati utilizzati nella documentazione presentata.

### Ambiente di esecuzione Java

L'ambiente di esecuzione (JRE, Java Runtime Environment) contiene la JVM ([Java Virtual Machine](#)), le librerie delle classi e un launcher per le applicazioni Java necessario per avviare i programmi scritti col linguaggio Java. La JRE non costituisce un ambiente di sviluppo software e non contiene tools di sviluppo.

### Autenticazione

Processo tramite il quale il software verifica la corretta, o almeno presunta, identità di un utente.

### Browser Web

Programma che consente agli utenti di visualizzare ed interagire con testi, immagini ed altre informazioni, tipicamente contenute in una pagina web di un sito.

### Client/Server

Rapporto tra due programmi in cui un programma, il client, fa una richiesta di servizio ad un altro programma, il server, che risponde a tale richiesta.

### DAO

(acronimo di *Data Access Object*). Una classe che rappresenta una entità tabellare di un database. Serve a stratificare e isolare l'accesso ad una tabella dalla parte della logica di business e quindi a creare un maggiore livello di astrazione.

### Database relazionale (tradotto in italiano con base di dati)

Archivio di dati, riguardanti uno stesso argomento o più argomenti correlati tra loro, strutturato in modo tale da consentire la gestione dei dati stessi (l'inserimento, la ricerca, la cancellazione ed il loro aggiornamento) da parte di applicazioni software.

### DTD

(acronimo di *Document Type Definition*). Ha lo scopo di definire le componenti ammesse nella costruzione di un documento [XML](#).

### Giocatore

L'utente che gioca all'applicazione software. Impersona il ruolo di sfidante o di banco a seconda delle regole del gioco.

### Java Server Pages

Tecnologia Java per lo sviluppo di applicazioni Web che forniscono contenuti dinamici in formato HTML o [XML](#).

### Java Virtual Machine

Macchina virtuale che consente l'esecuzione di programmi java compilati.

**Mano**

Indica il momento in cui sia il mazziere che il giocatore effettuano, ciascuno, una giocata. Una mano inizia quando il mazziere ha distribuito la carta coperta e termina quando il mazziere ha terminato il proprio turno.

**Matta**

Carta che può assumere il punteggio di una qualsiasi altra carta a discrezione del giocatore che la possiede. La matta può valere come una figura (mezzo punto) oppure un valore intero compreso tra uno e sette. Il re di denari svolge la funzione di matta.

**Mazziere** detto anche Banco

Colui che gestisce il mazzo e, distribuendo le carte agli altri giocatori, è l'ultimo a giocare.

**Posta**

Corrisponde alla vincita del giocatore. È pari alla puntata in caso di posta semplice; uguale al doppio della puntata in caso di posta doppia.

**Puntata**

La puntata è la quantità di denaro che si scommette all'inizio di ogni mano.

**Punteggio**

Corrisponde alla somma dei valori delle singole carte in possesso del giocatore.

**Sballare**

Situazione nella quale il giocatore si trova se il punteggio totalizzato supera il punteggio massimo (sette e mezzo). Si perde automaticamente la mano.

**Sette e Mezzo**

Punteggio massimo totalizzato che permette di rimanere in gioco.

**Sette e Mezzo Illegittimo**

Sette e mezzo realizzato con più di due carte. Può essere totalizzato solo dal banco.

**Sette e Mezzo Reale**

Detto anche Sette e Mezzo legittimo o Sette e Mezzo d'emblée, si ottiene quando il giocatore totalizza il punteggio massimo con due sole carte.

Le possibili combinazioni di carte che portano ad un Sette e Mezzo reale sono: sette ed una figura, matta ed una figura, matta ed un sette.

**Sistema Operativo**

Programma responsabile del controllo e della gestione dei componenti hardware che costituiscono un computer e dei programmi che su di esso girano.

**Stakeholder**

Soggetti portatori d'interessi nei confronti di un'iniziativa economica, sia essa un'azienda o un progetto.

**Stand Alone**

Sistema capace di funzionare da solo, indipendentemente dalla presenza di altri oggetti con cui potrebbe comunque interagire.

**Stare**

Rimanere in gioco senza chiedere ulteriori carte al mazziere.

**Web Server**

Processo, e per estensione il computer su cui è in esecuzione, che si occupa di fornire su richiesta del browser una pagina web.

**XML**

(acronimo di *eXtensible Markup Language*). È un linguaggio marcatore che consente la definizione di altri linguaggi marcatori. Viene utilizzato nei più disparati contesti: dalla definizione della struttura di documenti allo scambio di informazioni tra sistemi diversi, dalla rappresentazione di immagini alla definizione di formati di dati.