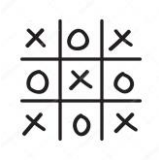
ESAME DI FONDAMENTI DI INFORMATICA T-2 del 6/2/2019
Proff. E. Denti – R. Calegari – G. Zannoni Tempo: 4 ore

NOME PROGETTO ECLIPSE: CognomeNome-matricola (es. RossiMario-0000123456)
NOME CARTELLA PROGETTO: CognomeNome-matricola (es. RossiMario-0000123456)
NOME ZIP DA CONSEGNARE: CognomeNome-matricola.zip (es. RossiMario-0000123456.zip)

NB: l'archivio ZIP da consegnare deve contenere l'intero progetto Eclipse

Si vuole sviluppare una semplice app per giocare a *Tris* (detto anche *TicTacToe,* v. figura a lato).



DESCRIZIONE DEL DOMINIO DEL PROBLEMA

Tris si gioca su una scacchiera 3x3 fra due giocatori, che utilizzano due simboli distinti (di norma X e O) per marcare le proprie mosse. I giocatori si alternano: vince il primo che riesce a porre tre suoi simboli in sequenza (orizzontale, verticale o diagonale). Se nessuno ci riesce, non ci sono vincitori e il gioco finisce in parità.

ENTITÀ COINVOLTE

Si desidera che ogni ***partita*** sia identificata in modo univoco dal timestamp (in formato ISO) reativo all'istante di inizio della partita stessa. A ogni partita è associata la lista delle configurazioni di ***scacchiera*** che la descrivono, a partire dalla scacchiera vuota iniziale fino alla configurazione finale in cui o uno vince (cosa che può avvenire anche prima che tutte le nove caselle siano state usate), o il gioco termina in parità. Ogni mossa genera quindi una nuova configurazione della scacchiera, che viene aggiunta in coda alla lista.

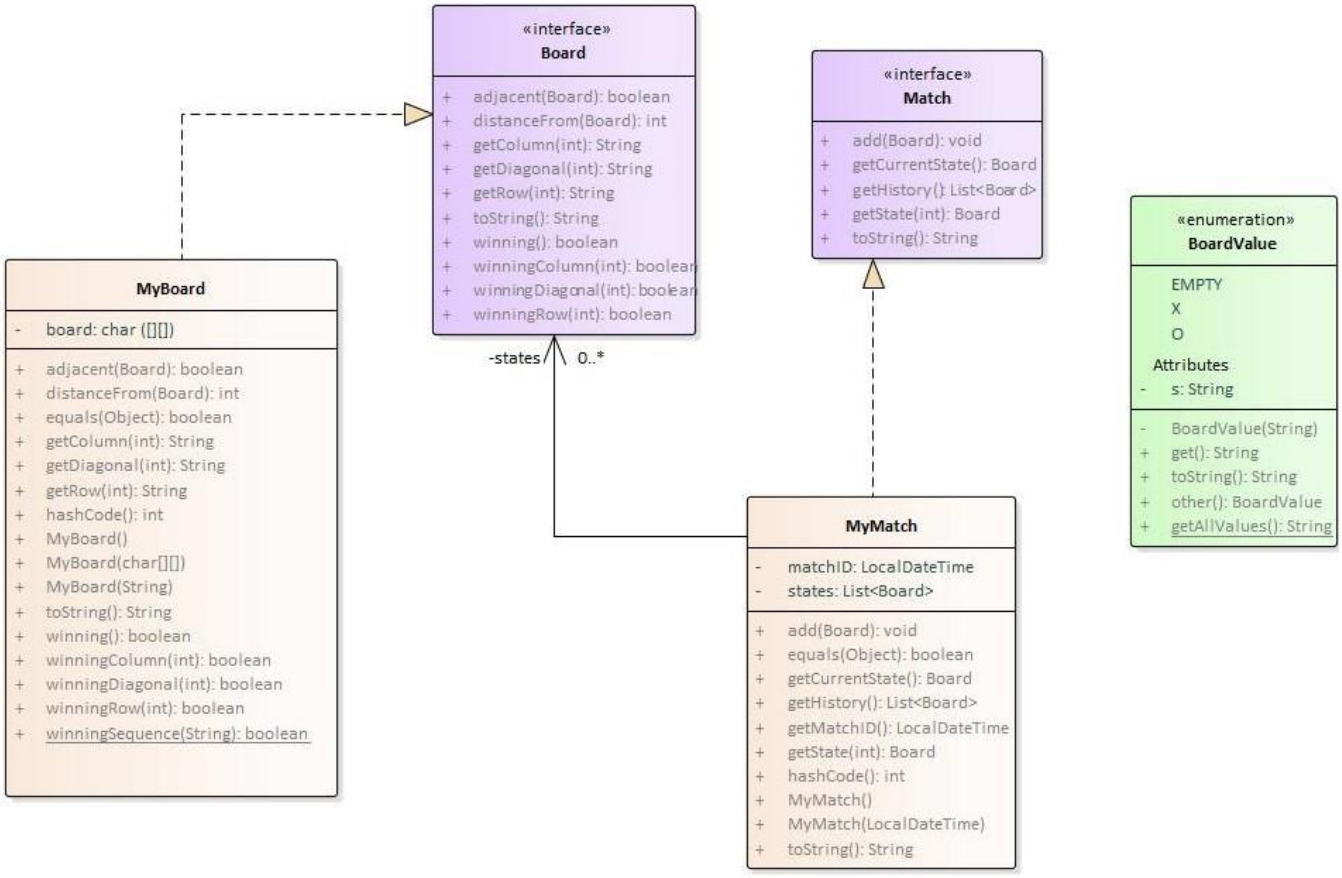
ALGORITMO DI GIOCO

Dopo ogni mossa occorre verifica l'eventuale vincita da parte di un giocatore controllando la presenza di tre simboli uguali in riga, colonna o diagonale.

Parte 1 (punti: 19)

Dati (namespace tris.model) *(punti: 17)*

Il modello dei dati deve essere organizzato secondo il diagramma UML di seguito riportato:

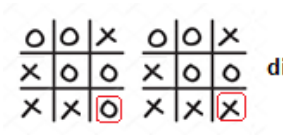


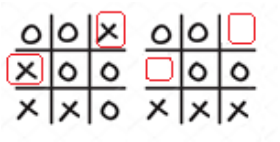
SEMANTICA:

a) L'enumerativo ***BoardValue*** (fornito) definisce i tre stati possibili per ogni singola cella della scacchiera (EMPTY, X, O): a ogni valore simbolico è associata la stringa corrispondente (per EMPTY si usa lo spazio " "), recuperabile sia tramite l'accessor ***get*** sia tramite ***toString***. Il metodo di utilità ***other*** restituisce invece l' "altro" simbolo rispetto all'attuale (ovviamente è indefinito nel caso il simbolo corrente sia EMPTY), mentre il metodo statico ***getAllValues*** restituisce la stringa coi tre valori possibili concatenati (ossia, " XO").

b) l'interfaccia ***Board*** (fornita) descrive la scacchiera, intesa come matrice 3x3 di ***BoardValue:*** per convenzione, righe e colonne sono numerate da 0 a 2, mentre le due diagonali sono numerate da 0 a 1. I metodi:

- ***getRow***, ***getColumn*** e ***getDiagonal*** consentono di recuperare rispettivamente la singola riga, colonna o diagonale di indice pari all'argomento ricevuto, restituendo la stringa corrispondente al contenuto (ad esempio, "XXO", "O O", etc.): lanciano ***IllegalArgumentException*** se l'indice è fuori range (ossia se non è compreso nell'intervallo 0-2 per ***getRow***, ***getColumn*** e 0-1 per ***getDiagonal***);
- analogamente ***winningRow***, ***winningColumn*** e ***winningDiagonal*** verificano rispettivamente se la riga, colonna o diagonale specificata (tramite il solito indice nel range 0-2 o 0-1, come sopra) sia vincente, ossia contenga tre simboli uguali;
- ***winning*** verifica se la scacchiera nel suo complesso sia vincente, ossia se in essa vi sia almeno una riga, colonna, o diagonale vincente;
- ***distanceFrom*** calcola la *distanza* fra la scacchiera corrente e quella fornita come argomento, intendendosi con ciò il numero di celle diverse fra le due configurazioni di scacchiera (vedi esempi illustrati di seguito);


distanza 1


distanza 2

- ***adjacent*** verifica se la scacchiera corrente sia *adiacente* a quella fornita come argomento, intendendosi con ciò che abbia distanza 1 da essa e inoltre che il numero di X e di O presenti nella scacchiera fornita come argomento sia quasi identico (più precisamente, che il numero di X differisca al più di 1 dal numero di O): ciò serve a evitare scacchiere "illegali" in cui lo stesso giocatore faccia due mosse consecutive
- ***toString*** deve restituire una stringa *adatta a visualizzare la scacchiera su console*, quindi con tre simboli per riga separati fra loro da tabulazioni e andando a capo dopo le prime due righe (ma non dopo l'ultima!). Per portabilità fra le piattaforme, il simbolo di "a capo" non dev'essere cablato come '\n' ma dev'essere espresso tramite ***System***.***lineSeparator***.

c) la classe **MyBoard** (da realizzare) implementa ***Board*** come da specifica, con le seguenti ulteriori precisazioni:

- devono essere previsti due costruttori, uno con argomento matrice 3x3 di caratteri, l'altro con argomento stringa di 9 caratteri consecutivi da intendersi letti dall'alto al basso e da sinistra a destra, senza separatori di alcun tipo: entrambi i costruttori lanciano ***IllegalArgumentException*** se l'argomento non ha la giusta dimensione o qualche carattere è diverso dai tre ammessi (quelli associati ai valori di ***BoardValue***)
- devono essere implementate anche ***equals*** e ***hashcode***: in particolare, per ***hashcode*** è sufficiente quella standard generata automaticamente da Eclipse (*menù Source → Generate hashcode and equals*) mentre ***equals*** deve considerare uguali due scacchiere *a distanza zero* una dall'altra;

d) l'interfaccia ***Match*** (fornita) descrive la partita intesa come sequenza di configurazioni di scacchiera. Metodi:

- ***getCurrentState*** restituisce lo stato attuale della scacchiera (che esiste sempre, perché la partita inizia con una scacchiera vuota, ossia con tutte le celle in stato EMPTY)
- ***getState*** restituisce lo stato i-esimo della scacchiera: se l'indice è fuori range, ossia supera quello dell'ultimo stato disponibile (quello corrente), viene lanciata ***IllegalArgumentException***

- ***getHistory*** restituisce la lista di tutti gli stati, dalla scacchiera vuota iniziale fino allo stato finale
- ***add*** aggiunge sequenza di configurazioni una nuova scacchiera, che diviene lo stato corrente, *purché il nuovo stato sia adiacente a quello precedente*: altrimenti, lancia ***IllegalArgumentException***
- ***toString*** deve restituire una stringa *adatta a visualizzare su console l'intera partita*, intesa come sequenza delle relative scacchiere separate fra loro da ***System.lineSeparator*** (non dopo l'ultima); anche qui il simbolo di "a capo" non dev'essere cablato come '\n' ma dev'essere espresso tramite ***System.lineSeparator***.

e) la classe **MyMatch** (da realizzare) implementa ***Match*** come da specifica, con le seguenti ulteriori precisazioni:

- devono essere previsti due costruttori, uno con argomento l'identificativo univoco ***LocalDateTime*** specificato, l'altro senza argomenti (che utilizza giorno e ora correnti). Entrambi devono inizializzare la lista delle configurazioni con la scacchiera iniziale vuota (ossia, con tutte le celle nello stato EMPTY)
- devono essere fornite due implementazioni standard di ***equals*** e ***hashcode*** che si suggerisce di far generare automaticamente da Eclipse (*menù Source → Generate hashcode and equals*)

Persistenza (namespace tris.persistence) *(punti: 2)*

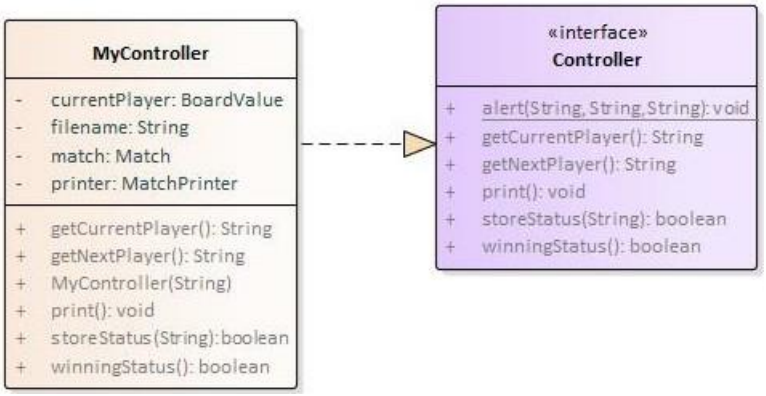
Obiettivo di questo componente è stampare su file una partita, secondo il diagramma UML di seguito riportato.

SEMANTICA:

a) l'interfaccia ***MatchPrinter*** (fornita) dichiara il solo metodo ***print*** che stampa un ***Match*** sul ***PrinterWriter*** fornito.

b) la classe **MyMatchPrinter** (da realizzare) implementa tale metodo, *assicurando che dopo ogni stampa il buffer di uscita venga opportunamente svuotato*.

Parte 2 (punti: 11)



Controller (namespace tris.controller)

Il controller – articolato in interfaccia e implementazione – è fornito già pronto: il suo stato interno mantiene lo stato della partita con annessi e connessi (giocatore attuale, etc.) nonché il printer da usare per le stampe. Conseguentemente, i suoi metodi fanno da ponte fra quelli del model e quelli del printer. In particolare:

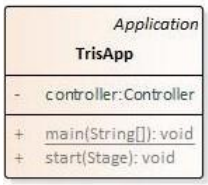
- ***storeStatus*** aggiunge alla partita corrente il nuovo stato di scacchiera ricevuto come argomento (sotto forma di stringa di 9 caratteri), verificando al contempo se esso sia vincente (restituisce a tal fine un boolean)
- ***winningStatus*** verifica se la partita corrente è vinta
- ***getCurrentPlayer*** restituisce la stringa corrispondente al giocatore corrente ("X" o "O")

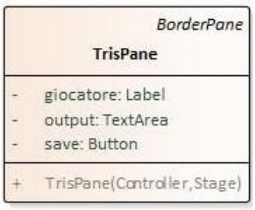
- ***getNextPlayer*** cambia il giocatore corrente (da "X" a "O", da "O" a "X") e restituisce la stringa corrispondente
- ***print*** stampa su file la partita corrente (che si suppone terminata)

L'interfaccia ***Controller*** offre altresì il metodo statico ***alert*** per far comparire una finestre di dialogo utile a segnalare errori all'utente (in questa applicazione, solo nel caso in cui la stampa su file fallisca per qualche motivo).

GUI (namespace tris.ui) *(punti: 11)*

L'interfaccia grafica dev'essere simile (non necessariamente identica) a quella sotto illustrata.





La classe ***TrisApp*** (fornita) contiene come sempre il main di partenza dell'applicazione.

La classe ***TrisPane*** (**pure fornita**) definisce il pannello con gli elementi grafici principali (Fig.1), ovvero:

- al top, l'etichetta con l'indicazione del giocatore corrente (ossia, se tocca a X o a O)
- a sinistra la griglia-scacchiera, sotto forma di istanza della classe ***TrisGrid***
- a destra, la textarea su cui far comparire via via lo stato della partita
- in basso, il pulsante *Stampa* (inizialmente disabilitato, si abilita solo quando la partita termina)

La classe ***TrisGrid*** (da realizzare) deve implementare la griglia e l'algoritmo di gioco, interfacciandosi opportunamente con ***TrisPane***. Più esattamente:

- il costruttore riceve da ***TrisPane*** tutti gli elementi su cui deve poter operare e alloca una griglia 3x3 di ***Button***, impostati con font *Courier New grassetto 20 punti* e inizializzati vuoti (ossia col testo corrispondente allo stato ***BoardValue.EMPTY***) (Fig. 1). SUGGERIMENTO: utilizzare un GridPane, ricordando però che il metodo add/4 prevede, in modo alquanto contro-intuitivo, *prima* l'indice di colonna, *poi* quello di riga.
  La pressione di uno dei 9 bottoni della griglia deve causare l'invocazione di un opportuno metodo di gestione eventi, per comodità denominato ***handle(Button)***
- il metodo ***toString*** fornisce la rappresentazione stringa dello stato attuale della griglia-scacchiera, nel formato di stringa a 9 caratteri consecutivi (adatto per il costruttore di ***Board***)
- quando il giocatore corrente preme uno dei 9 pulsanti della griglia-scacchiera di gioco, ***handle***:
  - imposta nel bottone premuto il simbolo del giocatore corrente (X o O)
  - disabilita il bottone stesso, in modo che non sia più cliccabile (cella già occupata)
  - aggiunge alla partita corrente, tramite il ***Controller*** (metodo ***storeStatus***), la nuova configurazione di scacchiera appena determinatasi, *verificando contemporaneamente* se qualcuno abbia vinto;
  - appende sulla textarea una nuova riga corrispondente alla nuova situazione della partita (Figg. 2,3,4): tale riga deve includere la configurazione della scacchiera in forma compatta (9 caratteri) e l'indicazione se il gioco continua o la partita è vinta (es. "continua" / "vittoria", "partita patta" o analogo);
  - se lo stato così ottenuto non è finale (vincente o patta) impone, tramite il Controller, di cambiare giocatore (metodo ***getNextPlayer***) e adegua la visualizzazione del giocatore nella label (Figg. 1,2,3,..);

- se, invece, lo stato così ottenuto è vincente o partita patta agisce di conseguenza (invocando un metodo di gestione ausiliario che per comodità chiameremo ***endGame***) e nell'ordine:
  - disabilita tutti i bottoni della griglia-scacchiera (perché la partita è finita)
  - abilita il pulsante *Stampa* (Fig. 4 o 5) associandolo al suo gestore eventi, così che, se premuto, esso effettui la stampa della partita (tramite **Controller.*print***) e subito dopo ri-disabiliti il pulsante *Stampa* **stesso** (Fig. 6).

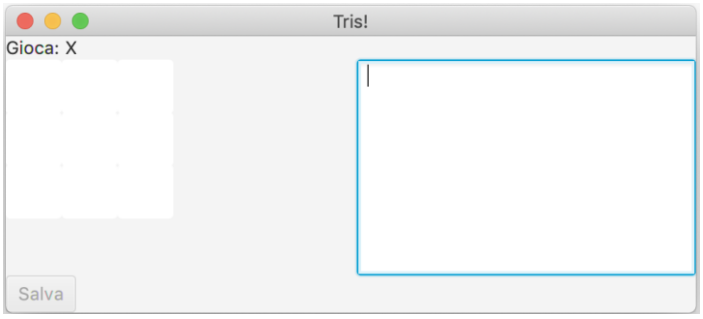


Figura 1

Figura 2

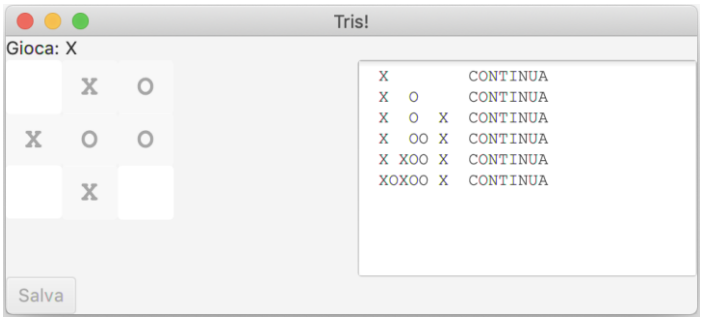


Figura 3

Figura 4

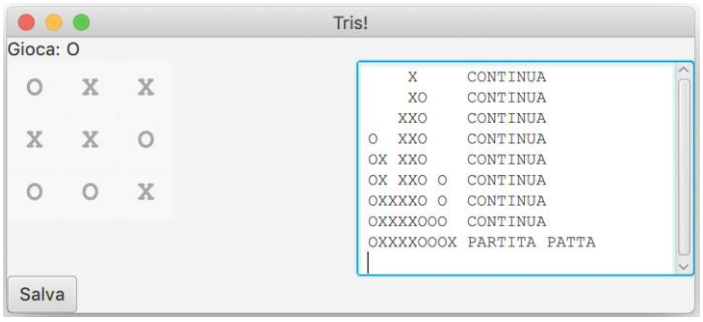


Figura 5

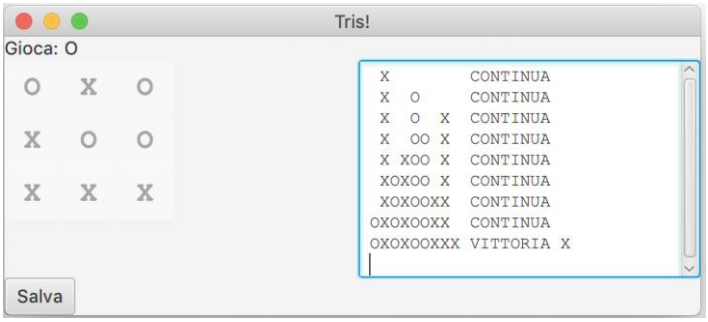


Figura 6