Appelli di Giugno, Luglio e Settembre 2005	♔♕♖♗♘♙ **Scacchi** ♚♛♜♝♞♟
	Laurea Triennale in Comunicazione Digitale Laboratorio di Informatica Generale

1 Descrizione

Il progetto consiste nel realizzare un programma per gestire scacchiere e scacchi e relative mosse. In particolare data una configurazione iniziale per la scacchiera, ed una sequenza di mosse del tipo da casella $c_{i,j}$ a casella $c_{k,h}$ vogliamo poter valutare la fattibilità di ogni singola mossa e aggiornare conseguentemente la scacchiera.

2 Struttura della Scacchiera

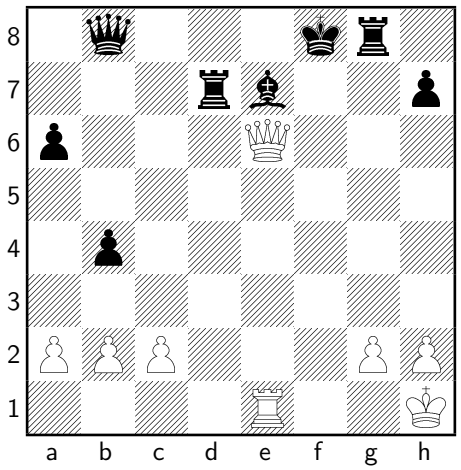
La scacchiera è quella classica del gioco degli scacchi, 8×8, le caselle bianche e nere si alternano, su questa sono disposti i pezzi della scacchiera.

La *configurazione* della scacchiera descrive la posizione dei pezzi in un preciso momento della partita.

La configurazione della scacchiera è codificata in una stringa nel seguente modo:

- \$ rappresenta il fine riga;
- un numero compreso tra 1 e 8 rappresenta quante caselle consecutive sono vuote (si parla di caselle consecutive disposte sulla stessa riga);
- le lettere K, Q, R, B, N e P rappresentano rispettivamente King (Re), Queen (Regina), Rook (Torre), Bishop (Alfiere), Knight (Cavallo) e Pawn (Pedina). Se minuscole sono pezzi neri, se maiuscole sono pezzi bianchi.

Pertanto la configurazione:



sarà rappresentata dalla stringa: `1q3kr1$3rb2p$p3Q3$8$1p6$8$PPP3PP$4R2K`.

3 Validità delle Mosse

Una mossa si riferisce sempre alla configurazione corrente ed è rappresentata da una quadrupla: i, j, k e h dove i e j rappresentano le coordinate (vedere la scacchiera dell'esempio precedente) della casella di partenza e k e h rappresentano le coordinate della casella di arrivo. Ovviamente i, j, k e h possono variare tra 1 e 8.

La validità di una mossa dipende dal pezzo presente sulla scacchiera nella casella indicata come casella di partenza. Ogni pezzo ha caratteristiche diverse e capacità di movimento differenti e come tale non tutte le caselle di arrivo sono lecite per tutti i pezzi. Le regole per il movimento dei vari pezzi sono le seguenti:

♚♔ **Re**, ha capacità di muoversi in tutte e otto (nord, nord-est, est, sud-est, sud, sud-ovest, ovest e nord-ovest) le direzioni, ma solo di una casella alla volta;

♛♕ **Regina**, ha capacità di muoversi in tutte e otto direzioni senza limite sul numero di caselle;

♜♖ **Torre**, ha capacità di muoversi solo in verticale od in orizzontale (nord, est, ovest e sud) senza limiti sul numero di caselle che compongono la mossa;

♝♗ **Alfiere**, ha capacità di muoversi solo in diagonale (nord-est, sud-est, sud-ovest e nord-ovest) senza limiti sul numero di caselle che compongono la mossa;

♞♘ **Cavallo**, si muove ad `'L'` per un totale di tre caselle: 2 in verticale e quindi 1 in orizzontale oppure 2 in orizzontale e quindi 1 in verticale;

♟♙ **Pedone** può muoversi secondo il seguente algoritmo:

- 2 caselle a nord se è la prima volta che si muove e la casella non è occupata;

- 1 casella a nord-est o a nord-ovest se la casella di arrivo è occupata da un pezzo dell'altro colore;
- 1 casella a nord se questa non è occupata.

Nota nord si scambia con sud se il pedone è dello schieramento nero.

Ovviamente se la casella di arrivo è occupata da un pezzo del colore avversario, questo verrà rimosso dalla scacchiera, se invece è occupata da un pezzo del nostro stesso colore la mossa non è possibile. Come semplificazione nel valutare le varie mosse non si considerano lo stato delle caselle lungo lo spostamento ma solo quella di arrivo.

4 Classi da Realizzare

È obbligatorio realizzare in Java il programma descritto nelle sezioni precedenti utilizzando le seguenti classi:

Piece

`Piece` è una classe astratta usata per descrivere il comportamento, in astratto, dei vari pezzi presenti sulla scacchiera.

Attributi

- **`char`** `color`: variabile di istanza che stabilisce il colore di appartenenza del pezzo corrispondente, `"W"` se bianco e `"B"` altrimenti.
- **`int`** `x` e **`int`** `y`: variabili di istanza che memorizzano la posizione corrente del pezzo sulla scacchiera.

Notare la peculiare presenza di alcuni attributi nella classe astratta, vale sempre la regola che lo stato degli oggetti deve essere nascosto ma in questo caso deve poter essere usato dalle classi concrete che estenderanno la classe astratta.

Metodi

- **`public abstract boolean`** `checkMove(`**`int`** `i, `**`int`** `j)`:
  verifica se il pezzo può muoversi nella posizione i,j. Se la mossa casella non può essere raggiunta dalla posizione corrente il metodo ritornerà **`false`**; se, invece, la mossa è legale e può essere fatta (non importa se la casella di arrivo è libera o occupata da un pezzo avversario) il metodo ritornerà **`true`**.
- **`public void`** `move(`**`int`** `i, `**`int`** `j)`:
  sposta il pezzo nella casella di posizione i,j senza curarsi della liceità della mossa.

King

La classe `King` estende la classe `Piece`.

- **`public`** `King(`**`int`** `x, `**`int`** `y, `**`char`** `c)` **`throws`** `IllegalPieceException`: costruttore che permette di creare un re. I parametri `x` e `y` rappresentano le coordinate della posizione che avrà il pezzo nella configurazione corrente della scacchiera, mentre il parametro `c` rappresenta il colore dello schieramento di appartenenza del pezzo che si sta creando. Se le coordinate determinano una casella fuori dalla scacchiera o il carattere passato come parametro non rappresenta uno dei colori ammissibili viene sollevata un'eccezione di tipo `IllegalPieceException` il cui messaggio spiega le condizioni di errore.

- **`public boolean`** `checkMove(`**`int`** `i, `**`int`** `j)`: implementazione del corrispondente metodo astratto di `Piece`. La semantica dipende dalla libertà di movimento del pezzo e la trovate nella corrispondente sezione.

Si dovranno anche creare le classi `Queen`, `Rook`, `Bishop`, `Knight`, e `Pawn` che analogamente implementeranno i corrispondenti pezzi degli scacchi.

ChessBoard

Descrive la scacchiera in termini di una sua configurazione.

Attributi

- `Piece[][] chessboard`: variabile di istanza che memorizza la configurazione corrente dei pezzi sulla scacchiera. Ogni elemento conterrà un pezzo oppure null se in quella posizione non ci sono pezzi nella configurazione corrente.

- `String filename`: variabile di istanza contenente il nome del file su cui si salvano le configurazioni e che viene usato per il ripristino della configurazione. Implementare anche il corrispondenti metodi publici `String getFilename()` e **`void`** `setFilename(String)` dall'ovvio significato.

Metodi e Costruttori

- **`public`** `ChessBoard(String filename)`: costruttore che configura la scacchiera, `filename` è il nome del file su disco contenente il una serie di configurazioni, una per riga, l'ultima configurazione letta diventerà la configurazione corrente.

- **`public`** `String toString()`: metodo che crea e ritorna una stringa rappresentante la configurazione corrente della scacchiera, sfruttando le convenzioni descritte precedentemente;

- **`public void`** `setChessBoard(String)`: metodo che permette di inizializzare la scacchiera alla configurazione passata come parametro, la configurazione è codificata nella stringa secondo le stesse convenzioni descritte precedentemente;

- **public** `String getChessBoard()`: metodo che ritorna la configurazione corrente codificata secondo le convenzioni descritte precedentemente;
- **public void** `write()`: metodo che appende la configurazione corrente della scacchiera al file corrispondente;
- **public void** `reload()`: metodo che rilegge dal file indicato dall'attributo `filename` la configurazione della scacchiera;
- **public** `Object clone()`: metodo che crea una copia dell'oggetto e la ritorna;
- **public void** `backup(String)`: metodo che effettua una copia di backup della configurazione della scacchiera, salvandola in un file il cui nome è specificato nella stringa passata come argomento, il file verrà creato ex-novo;
- **public boolean** `checkMoves(String moves)` **throws** `IllegalMoveException`: metodo che controlla che una sequenza di mosse siano tutte mosse valide a partire dalla configurazione corrente della scacchiera. Il parametro `moves` contiene la sequenza di mosse da controllare, le mosse sono codificate come segue:

  $$\mathrm{m}_1^i\mathrm{m}_1^j\mathrm{m}_1^k\mathrm{m}_1^h;\mathrm{m}_2^i\mathrm{m}_2^j\mathrm{m}_2^k\mathrm{m}_2^h;\cdots;\mathrm{m}_n^i\mathrm{m}_n^j\mathrm{m}_n^k\mathrm{m}_n^h\$$

  In questa sequenza sono codificate n mosse:

  - ogni mossa è rappresentata da una quadrupla $\mathrm{m}_i^i\mathrm{m}_i^j\mathrm{m}_i^k\mathrm{m}_i^h$ con $i = 1,\ldots,n$;
  - i singoli m_i^r con $r \in \{i,j,k,h\}$ e $i = 1,...,n$ sono interi compresi tra 1 e 8 e rappresentano ascisse e ordinate di caselle della scacchiera;
  - le mosse sono separate da `";"` e non ci sono spazi aggiuntivi;
  - la sequenza di mosse è terminata da un `"$"`.

  Se tutte le mosse sono legali, il metodo restituirà **`true`** e la nuova configurazione corrente sarà il risultato dell'applicazione delle varie mosse. Attenzione alla rimozione dei pezzi mangiati durante le varie mosse.

  Se una mossa si rivela legale ma la casella di arrivo è occupata da un pezzo dello stesso schieramento, la valutazione si interrompe, il metodo ritornerà **`false`** e la configurazione corrente sarà la configurazione ottenuta applicando le mosse fino alla mossa inattuabile.

  Se una mossa si rivela illegale si ripristina la configurazione iniziale della scacchiera e si solleva un'eccezione di tipo `IllegalMoveException` il cui messaggio specifica qual è la mossa illegale. Sulla legalità delle mosse vedere la sezione apposita.
- **public** `String compareChessBoards(ChessBoard)`: metodo che confronta due configurazioni e stabilisce se dalla prima configurazione (scacchiera chiamante) è possibile in una mossa arrivare nella configurazione passata come parametro. In altre parole controlla se esiste una mossa ammissibile che permette di passare dalla configurazione chiamante alla configurazione passata come parametro. Se

la mossa esiste il metodo ritornerà una stringa che rappresenta tale mossa come descritto per il metodo `checkMoves()` (senza `";"` in fondo) altrimenti ritornerà la stringa vuota.

Eccezioni.

`InvalidMoveException` e `IllegalPieceException`, da definire in modo opportuno, che rappresentano le eccezioni da lanciare (con le modalità descritte) quando si verificano le condizioni di errore descritte nei punti precedenti.

Tutte le altre eccezioni previste dall'uso di metodi Java devono filtrare ed essere gestite nel metodo `main()` anche se non espressamente indicato dalla segnature dei metodi introdotti in questo documento.

Game

La classe `Game` rappresenta la classe da lanciare per eseguire il progetto. Questa classe contiene esclusivamente il metodo `main()`, che permette di:

- leggere (o rileggere) la configurazione da file;
- visualizzare (come stringa) la configurazione corrente;
- introdurre delle mosse da verificare;
- costruire un'altra scacchiera con cui confrontare la configurazione corrente.

A parte quanto espressamente richiesto, è lasciata piena libertà sull'implementazione delle singole classi e sull'eventuale introduzione di altre classi, a patto di seguire le regole del paradigma ad oggetti ed i principi di buona programmazione.

Non è richiesto e non verrà valutato l'utilizzo di particolari modalità grafiche di visualizzazione: è sufficiente una qualunque modalità di visualizzazione basata sull'uso dei caratteri.

È invece espressamente richiesto di non utilizzare package non standard di Java (si possono quindi utilizzare `java.util`, `java.io` e così via), con l'unica eccezione del package `prog.io` incluso nel libro di testo per gestire l'input da tastiera e l'output a video e di rispettare l'interfaccia fornita.

5 Modalità di Consegna

Il progetto deve essere svolto a gruppi di al massimo tre persone che intendono sostenere l'intero esame di Informatica Generale e Laboratorio negli appelli di Giugno, Luglio o Settembre 2005, e deve essere consegnato entro mezzanotte di domenica 12 giugno 2005, utilizzando il sito di sottoposizione delle esercitazioni (all'indirizzo `http://infogen.dsi.unimi.it`). Per poter effettuare la sottoposizione è necessario autenticarsi utilizzando un nome di login e una password. Nella pagina principale

del sito stesso è spiegato come ottenere questi dati. Nel caso il progetto venga svolto da più di una persona, dovrà essere fatta in ogni caso una sola sottoposizione, indicando chiaramente in un commento all'inizio dei sorgenti consegnati nome, cognome e matricola dei vari componenti del gruppo.

Dovranno essere consegnati tutti i sorgenti Java che permettano al programma di essere eseguito correttamente, compressi in un archivio di tipo ZIP che estragga i file nella directory in cui si trova l'archivio stesso (altri tipi di sottoposizioni verranno automaticamente rifiutate dal sito). Nell'archivio dovrà anche essere accluso un breve documento in formato txt o rtf in cui:

- verrà descritto il modo in cui interfacciarsi con il programma;
- saranno illustrate le principali scelte implementative e le strategie utilizzate per svolgere il progetto

Il sistema rifiuterà automaticamente le sottoposizioni i cui sorgenti contengano errori rilevati in fase di compilazione.

È inoltre richiesto di consegnare una copia cartacea della stampa del codice sorgente prodotto in portineria del DSI indicando chiaramente nome, cognome e numero di matricola di tutti i componenti del gruppo, nonché il turno e il docente di riferimento. Nella copia cartacea indicare anche in quale appello il gruppo intende discutere il progetto.

6 Valutazione

Durante la prova orale con i singoli studenti saranno discusse le modalità implementative adottate e la padronanza di alcuni dei concetti necessari per preparare il progetto e/o spiegati a lezione. La valutazione del progetto sarà fatta in base alla:

- conformità dell'implementazione scelta per risolvere il problema con il paradigma di programmazione a oggetti;
- conformità del codice presentato alle regole di buona programmazione;
- adeguatezza del manuale utente presentato a descrivere il modo in cui un utente può utilizzare il programma;
- assenza di errori nel programma;
- usabilità del programma;

Walter Cazzola
Dipartimento di Informatica e Comunicazione
Via Comelico 39/41 20135 Milano
Stanza S233 — Tel. +39.010.353.6637
e-mail: cazzola@dico.unimi.it

Dario Malchiodi
Dipartimento di Scienze dell'Informazione
Via Comelico 39/41 20135 Milano
Stanza S238 — Tel. +39.02.503.16338
e-mail: malchiodi@dsi.unimi.it