

Corso di Programmazione e Algoritmi 2
Modulo di Laboratorio
A.A. 2023/2024

Regole per lo svolgimento del progetto:

- Il progetto può essere svolto **singolarmente** o **in gruppi di qualsiasi dimensione**.
In ogni caso, ciascuno studente dovrà poi **singolarmente** discutere il progetto dimostrando di essere in grado di modificare e/o aggiungere alcune parti. In caso di esito sufficiente di tale discussione, **verrà attribuito il punteggio da 0 a 3 punti** che è additivo rispetto al voto della parte da 6 CFU di teoria.
- Il progetto deve essere consegnato **almeno 7 giorni prima** della prova orale di *Programmazione e Algoritmi 2* relativa all'appello in cui si desidera discutere il progetto. Non è necessario che la restante parte dell'esame (da 6 CFU) sia svolta nel medesimo appello.
- Il progetto deve contenere le classi pubbliche e i metodi richiesti **rispettandone esattamente il nome, il tipo e l'ordine dei parametri formali, ed il tipo di ritorno**. Si tenga presente che può essere necessario sviluppare anche altre classi (non pubbliche) oltre quelle richieste.
- Si tenga presente che **non sono presenti** tutti i campi di istanza o i modificatori (***public, private, protected, static, final, ...***) che devono essere opportunamente aggiunti da voi nella consegna.
- I **metodi di accesso e modifica** dei vari parametri non sono tutti presenti nella specifica. Deve essere vostra cura aggiungerli, dove occorrono, in modo opportuno, così come il metodo ***equals***.
- Si è naturalmente liberi di sviluppare (e anzi siete incoraggiati a farlo) classi e/o metodi aggiuntivi, laddove lo si ritenga utile o necessario.
- Deve essere consegnato via email all'indirizzo luca.moscardelli@unich.it un file compresso contenente tutti i file sorgenti e avente come nome
[java] progetto 2023_2024.zip
l'email deve avere come oggetto
[java] consegna progetto programmazione 2023/2024
e deve contenere nel testo la lista (Cognome, Nome, Matricola e Email) di tutti gli studenti che hanno svolto il progetto consegnato.
In caso di consegna in gruppo, la consegna deve essere effettuata da uno dei componenti del gruppo, mettendo in CC nell'email anche l'indirizzo di tutti gli altri componenti.
- Non è necessario consegnare il metodo ***main*** (che voi potete chiaramente utilizzare per i vostri test)
- **L' unica libreria del Java che potete è `java.util.ArrayList`** (oltre naturalmente alle classi del package **`java.lang`** e ai file forniti insieme a questa specifica con la libreria grafica EZ)
- Le classi del package **`cleii.ez`** non devono essere modificate e non vanno incluse nel file zip della consegna.

Testo del progetto:

Sviluppare in **Java** un package **cleii.campominato** per la rappresentazione ed esecuzione di partite del famoso videogioco solitario “campo minato”. Per la descrizione del gioco si faccia riferimento alla pagina Wikipedia [https://it.wikipedia.org/wiki/Campo_minato_\(videogioco\)](https://it.wikipedia.org/wiki/Campo_minato_(videogioco)).

Al link <https://g.co/kgs/2jsvCv> è invece disponibile una versione giocabile online del gioco.

Si noti che una casella del gioco può avere fino a 8 caselle adiacenti (in direzione N, S, E, W, NE, NW, SE, SW).

Impostazione formale

Il package deve contenere le seguenti **classi pubbliche**:

- **Tabellone**

Classe pubblica che rappresenta un tabellone del gioco.

Contiene i seguenti metodi:

- Costruttore

`public Tabellone(Partita p, int larghezza, int altezza)`
che costruisce un tabellone, relativo alla partita `p`, avente altezza `righe` e larghezza `colonne`.

- Costruttore “di copia”

`public Tabellone(Tabellone t)`
che costruisce un tabellone che è la copia del tabellone `t`. Attenzione: le modifiche eventualmente fatte al nuovo tabellone non devono ripercuotersi su `t`.

- `public void segnaCasella(int riga, int colonna)`

che contrassegna la casella di riga `riga` e colonna `colonna` come contenente una mina, se non era già stata contrassegnata tale; altrimenti toglie il contrassegno dalla casella stessa.

- `public boolean isSegnata(int riga, int colonna)`

che restituisce `true` se e solo se la casella di riga `riga` e colonna `colonna` è stata contrassegnata come casella contenente una mina, `false` altrimenti.

- `public void scopriCasella(int riga, int colonna)`

che segna come scoperta la casella di riga `riga` e colonna `colonna`.

- `public boolean isCoperta (int riga, int colonna)`

che restituisce `true` se e solo se la casella di riga `riga` e colonna `colonna` è coperta nel tabellone.

- `public String toString()`
che deve restituire una stringa che rappresenta il tabellone corrente secondo la seguente convenzione:
 - C corrisponde a una casella **coperta** che **non contiene mine** e **non è segnata come contenente mine**
 - D corrisponde a una casella **coperta** che **non contiene mine** ed è **segnata come contenente mine**
 - X corrisponde a una casella **coperta** che **contiene mine** e **non è segnata come contenente mine**
 - Y corrisponde a una casella **coperta** che **contiene mine** ed è **segnata come contenente mine**
 - Z corrisponde a una casella **scoperta** che **contiene mine** (ce ne può essere solo una, poiché appena si scopre una casella con una mina il gioco termina)
 - Le cifre da 0 a 8 corrispondono a una casella **scoperta non contenente mine**, con la cifra che indica il numero di caselle adiacenti contenenti mine.

Ad esempio,

```
CCCCCCCCC
CCCXCCCCC
CCCCCCCCXC
CCCCXXCCXC
CCXCCCCCCC
CCCCCCCXCC
```

corrisponde al tabellone relativo alla configurazione iniziale di una partita con larghezza 10, altezza 6 e 7 mine (in cui nessuna casella è segnata);

```
CCCCCCCCC
CCCXCCCCC
CCCCCCCCXC
CCCCXXCCXC
CCX2222CCC
CCC1001XCC
```

corrisponde al tabellone relativo alla configurazione ottenuta a partire da quella del precedente esempio dopo che è stata scoperta la casella di riga 6 e colonna 5, e in cui nessuna casella è segnata.

- **Partita**

Classe pubblica che rappresenta una partita del gioco.

Contiene i seguenti metodi:

- **Costruttore**
`public Partita (int larghezza, int altezza, int nMine)`
che costruisce una partita del gioco contraddistinta da un tabellone avente `altezza` righe e `larghezza` colonne, nel quale sono posizionate in modo casuale `nMine` mine. Lo stato della partita è impostato su **partita in corso**;
- **Metodo di accesso** `public int getLarghezza()`
- **Metodo di accesso** `public int getAltezza()`
- **Metodo di accesso** `public int getNMine()`
- `public void segnaCasella(int riga, int colonna)`
che, se lo stato della partita è **partita in corso**, contrassegna la casella di riga `riga` e colonna `colonna` come contenente una mina, se non era già stata contrassegnata tale; altrimenti toglie il contrassegno dalla casella stessa.
Se lo stato della partita non è **partita in corso**, viene lanciata l'eccezione `AzioneNonValida`.
- `public boolean getCasellaSegnata(int riga, int colonna)`
che restituisce `true` se e solo se, nel tabellone della configurazione corrente, la casella di riga `riga` e colonna `colonna` è stata contrassegnata dal giocatore come casella contenente una mina, `false` altrimenti.
- `public void scopriCasella(int riga, int colonna)`
che, se lo stato della partita è **partita in corso**, scopre la casella di riga `riga` e colonna `colonna`, e di conseguenza
 - se è stata scoperta una casella che non contiene una mina: se tale casella è **sicura** (cioè non ha nessuna casella adiacente contenente mine), vengono scoperte tutte le caselle adiacenti, iterando il procedimento fino a che tutte le caselle sicure hanno tutte le caselle a loro adiacenti scoperte. La partita viene portata in stato di **terminazione con successo** se dopo aver scoperto la casella di riga `riga` e colonna `colonna` (ed eventualmente le caselle adiacenti alle caselle sicure) rimangono da scoprire solo caselle che contengono mine.
 - se è stata scoperta una casella contenente una mina porta la partita in stato di **terminazione senza successo**.

Al termine, viene opportunamente memorizzato il tabellone così ottenuto in modo che possa essere in seguito restituito dal metodo `getEvoluzione`.

Se lo stato della partita non è **partita in corso**, viene lanciata l'eccezione `AzioneNonValida`.

- `public int getMineAdiacenti(int riga, int colonna)`
che restituisce il numero di caselle adiacenti contenenti mine (0 se la casella è sicura).
- `public boolean contieneMina(int riga, int colonna)`
che restituisce `true` se e solo se la casella di riga `riga` e colonna `colonna` contiene una mina.
- `public boolean isCoperta (int riga, int colonna)`
che restituisce `true` se e solo se la casella di riga `riga` e colonna `colonna` è coperta nel tabellone della configurazione corrente.

- `public int getStatoPartita()`
che restituisce 0 se la partita è in corso, 1 se la partita è terminata con successo e 2 se la partita è terminata senza successo.
- `public ArrayList<Tabellone> getEvoluzione()`
che restituisce un ArrayList contenente il tabellone iniziale e i tabelloni ottenuti dopo ogni mossa (che consiste nello scoprire una casella) del giocatore.
- `public String toString()`
che deve restituire una stringa che contiene, in sequenza, le rappresentazioni di tutti i tabelloni ottenuti a seguito del fatto che il giocatore scopre delle caselle, preceduta dall'indicazione della mossa (che consiste nello scoprire una casella) che ha condotto a tale tabellone:

Tabellone iniziale:

```
CCCCCCCCC
CCCXCCCCC
CCCCCCCCXC
CCCCXXCCXC
CCXCCCCCCC
CCCCCCCXCC
```

Tabellone a seguito della mossa di riga 6 e colonna 5:

```
CCCCCCCCC
CCCXCCCCC
CCCCCCCCXC
CCCCXXCCXC
CCX2222CCC
CCC1001XCC
```

- **AzioneNonValida**
Classe pubblica che rappresenta l'eccezione da sollevare come descritto nella classe `Partita`.

○ **InterfacciaGrafica**

Classe pubblica che implementa l'interfaccia `cleii.ez.Ascoltatore`

L'interfaccia grafica deve sfruttare unicamente le classi del package `cleii.ez`.

La richiesta minimale è che venga disegnato il tabellone in modo che alle diverse tipologia di casella (disegnate come quadrati) siano associati diversi colori:

tipologia casella	Colore nella grafica
Casella coperta non segnata	Grigio chiaro
Casella coperta segnata	Rosso
Casella scoperta (senza mine) sicura	Bianco
Casella scoperta (senza mine) con $n > 0$ mine adiacenti	Bianco con n visualizzato
Casella scoperta con mina	Nero

L'interfaccia con l'utente avviene attraverso mouse e tastiera:

- il **click con il tasto sinistro** su una casella provoca l'azione di scoprire la casella (se la casella è già scoperta non avviene nulla).
- il **click con il tasto destro** segna la casella come contenente mine (se non è già segnata) o rimuove tale segno (se è già segnata).
- La pressione del tasto **Esc** della tastiera provoca la visualizzazione di tutte le mine e inibisce il proseguimento della partita.

Contiene (tra gli altri) il seguente metodo:

○ **Costruttore**

`public InterfacciaGrafica (Partita p)`

che costruisce e visualizza una interfaccia grafica per la partita p , visualizzando come punto di partenza il tabellone della configurazione corrente della partita e permettendo l'interazione con l'utente a partire da tale configurazione. Può quindi essere passata come parametro sia una partita appena iniziata, in cui il tabellone ha tutte le caselle coperte, sia una partita in cui sono già state scoperte e/o segnate alcune caselle.