ESAME DI FONDAMENTI DI INFORMATICA T-2 del 16/06/2020

Proff. E. Denti – R. Calegari – A. Molesini

Tempo a disposizione: 3 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)
NOME JAR DA CONSEGNARE: CognomeNome-matricola.jar (es. RossiMario-0000123456.jar)

Si devono consegnare DUE FILE: *l'intero progetto Eclipse* e *il JAR eseguibile*

Si ricorda che compiti *non compilabili* o *palesemente lontani da 18/30* NON SARANNO CORRETTI e causeranno la verbalizzazione del giudizio "RESPINTO"

È stata richiesta una app per agevolare la soluzione di ***Crittocruciverba*** (detto anche cruciverba crittografato o cruciverba cifrato).

DESCRIZIONE DEL DOMINIO DEL PROBLEMA

Il ***crittocruciverba*** è una tipologia di gioco enigmistico in cui il solutore deve risolvere lo schema non già partendo da definizioni di parole (che non sono fornite), ma in base al principio "*a numero uguale corrisponde lettera uguale*".

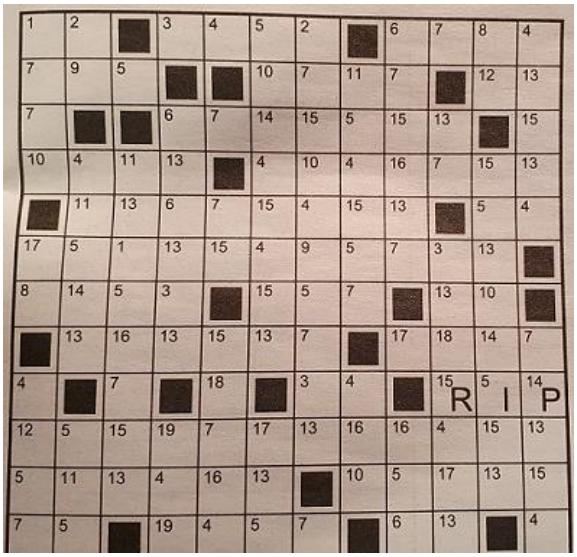
A tal fine, ogni casella dello schema (escluse ovviamente le caselle nere) è numerata: per iniziare il gioco, alcune caselle – di solito tre o quattro - contengono già le rispettive lettere.

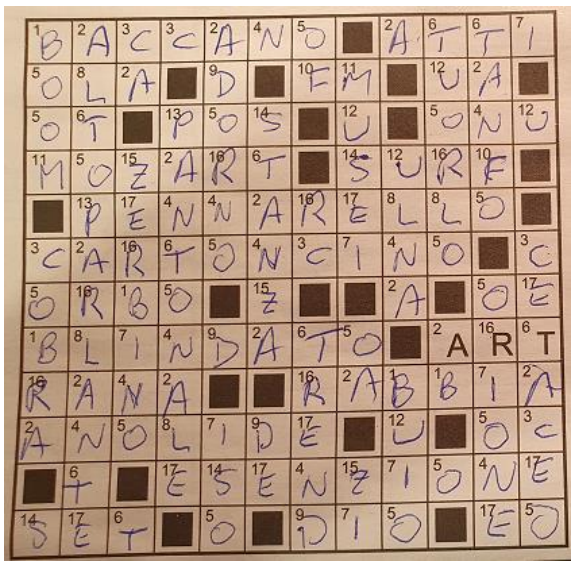
ESEMPIO (foto): nello schema a lato, la "R" corrisponde al 15, la "I" al 5 e la "P" al 14.

Il solutore inizia quindi riportando preliminarmente tali lettere in tutte le caselle che hanno gli stessi numeri.

ESEMPIO: occorrerà quindi scrivere "R" in tutte le caselle etichettate dal numero 15, "I" in tutte le caselle etichettate dal 5, e "P" in tutte quelle etichettate dal 14.

Terminata tale fase preliminare, il solutore deve indovinare le altre lettere basandosi sul proprio intuito, solitamente cercando di capire la posizione delle vocali, o iniziando da parole di cui conosca già alcune lettere e cercando di identificare le altre. Chiaramente, il gioco termina quando il solutore riesce e risolvere l'intero schema.



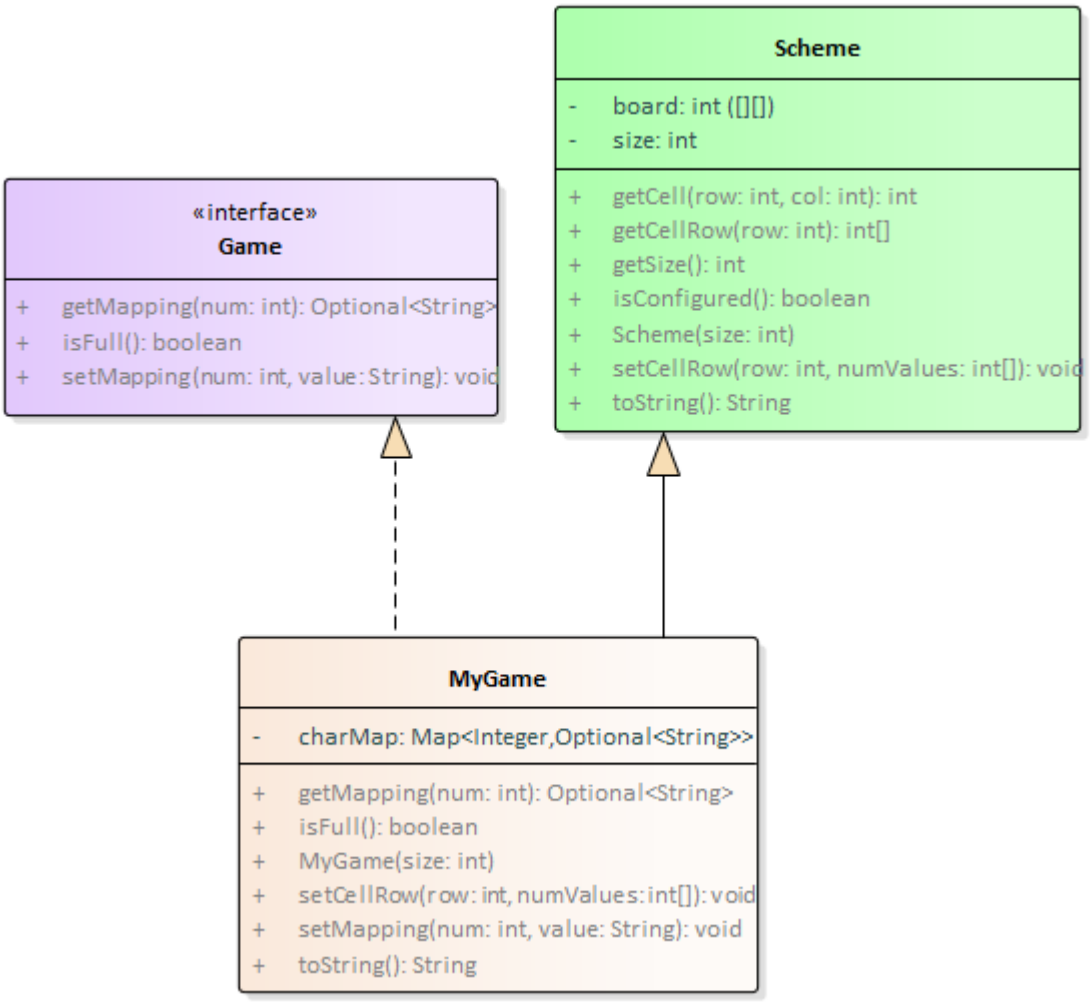


Il crittocruciverba da risolvere è descritto in un apposito file di testo, il cui formato è riportato più oltre.

TEMPO STIMATO PER SVOLGERE L'INTERO COMPITO: 1h50 – 2h20

Parte 1 (punti: 22)

Modello dei dati (package crosswords.model) **[TEMPO STIMATO: 1h-1h10 minuti]** *(punti: 12)*



SEMANTICA:

a) la classe ***Scheme*** (fornita) rappresenta lo schema di gioco statico, inteso come *scacchiera quadrata* di valori interi, senza alcuna lettera associata. *Per convenzione, la casella nera è rappresentata dal valore 0.* Il costruttore riceve la dimensione della scacchiera e inizializza tutte le celle al valore -1, che rappresenta convenzionalmente la casella non ancora inizializzata. Per questo il metodo *isConfigured* restituisce *true* se e solo se non c'è alcuna cella di valore -1. L'altro metodo fondamentale è *setCellRow*, che consente di caricare nello schema un'intera riga di valori. Completano la classe ovvi metodi accessor e *toString*: quest'ultima emette una stringa con la struttura della scacchiera (in righe) intesa come sequenza dei numeri che etichettano le varie caselle, *utilizzando il carattere "0" per indicare le caselle nere*.

b) L'interfaccia ***Game*** (fornita) rappresenta lo schema di gioco dinamico, ossia con le lettere associate ai numeri. A tal fine dichiara i seguenti metodi:

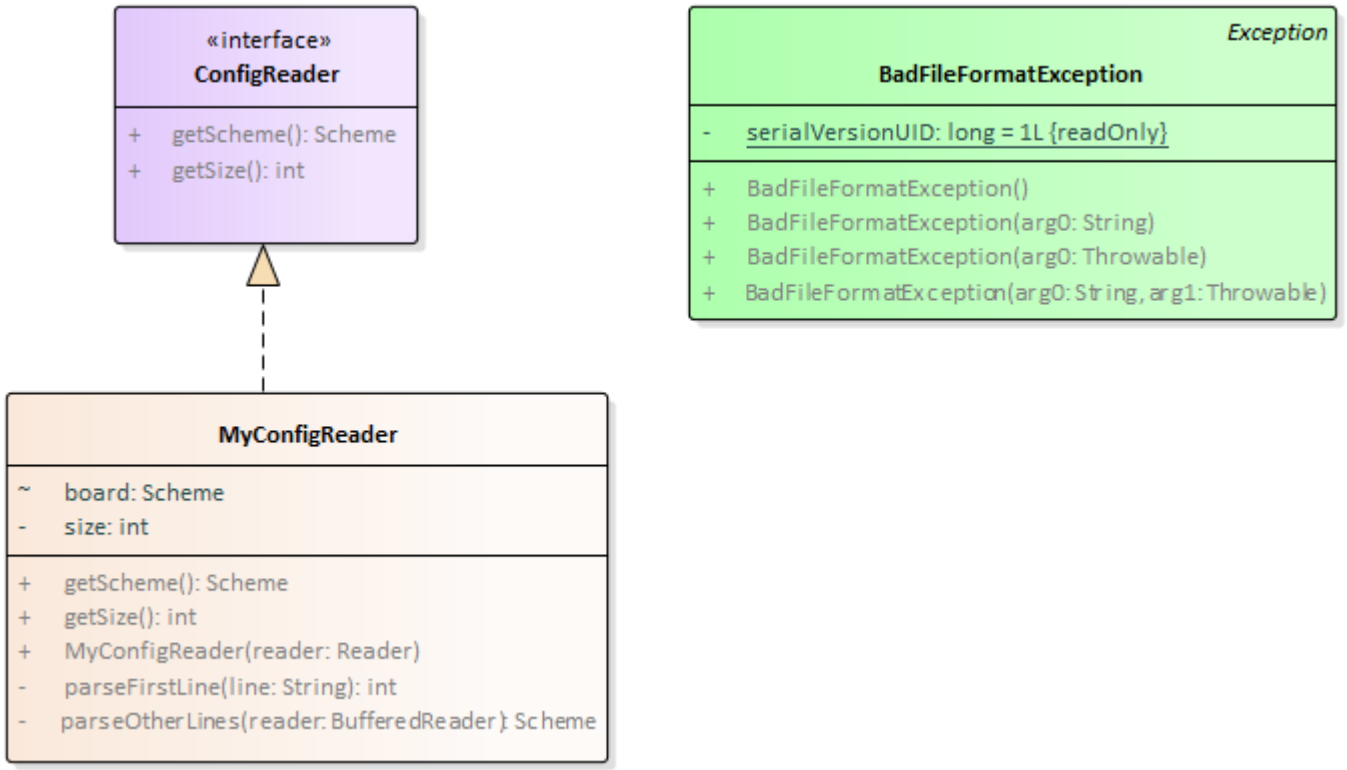
- *isFull* che restituisce *true* se e solo se tutte le celle hanno una lettera associata (e il gioco è finito)
- *setMapping* che inserisce un mapping fra intero e stringa
- *getMapping* che estrae il mapping fra intero e stringa, dato l'intero

c) la classe ***MyGame*** (**da realizzare**) estende ***Scheme*** e implementa ***Game*** come sopra specificato: *si suggerisce di rappresentare le lettere come stringhe* (optional), eventualmente utilizzando internamente una mappa Map<Integer,Optional<String>>. Il costruttore riceve, come in ***Scheme***, la dimensione della scacchiera.
Il metodo *toString* deve emettere una stringa con il contenuto della scacchiera (in righe), *utilizzando il carattere "#" per indicare le caselle nere*. Quindi, a differenza della classe ***Scheme***, se la cella è associata a

una stringa viene mostrata la stringa anziché il numero: altrimenti viene mostrata, come in ***Scheme***, l'etichetta numerica della cella stessa

NB: a seconda dell'organizzazione interna che si adotta, può essere necessario adattare il metodo *setCellRow*

Persistenza (crosswords.persistence) **[TEMPO STIMATO: 30-40 minuti]** *(punti 10)*



Lo schema di gioco è descritto nel file di testo **config.txt** formattato come segue:

- la prima riga contiene la parola DIM seguita, dopo uno spazio, dalla dimensione nella forma NxN, dove N è un intero e il carattere intermedio è una "x". Da notare che la scacchiera è *sempre quadrata*.
- le righe successive descrivono la scacchiera, per righe: i valori numerici sono separati fra loro da spazi, mentre le celle nere sono rappresentate dal carattere "#"

Ad esempio:

```
DIM 12x12
1 2 3 4 5 3 6 6 7 # 8 2
9 10 # 6 3 8 # 11 12 2 4 7
...
```

N.B: a differenza di quanto riportato nella descrizione iniziale del dominio, non ci sono lettere di aiuto iniziali!!!

SEMANTICA:

a) L'interfaccia ***ConfigReader*** (fornita) dichiara i due metodi *getSize* e *getScheme*

b) La classe ***MyConfigReader*** (**da realizzare**) implementa ***ConfigReader***: il costruttore riceve un Reader già aperto ed effettua la lettura, costruendo e memorizzando internamente lo ***Scheme*** corrispondente. I due accessor specificati da ***ConfigReader*** restituiscono rispettivamente tale ***Scheme*** (metodo *getScheme*) e la relativa dimensione (metodo *getSize*). In caso di problemi di I/O deve essere propagata l'opportuna ***IOException***, mentre in caso di problemi di formato dei file deve essere lanciata una ***BadFileFormatException*** (fornita) il cui messaggio dettagli l'accaduto.
In particolare, il reader deve verificare: 1) che la prima riga rispetti rigorosamente il formato previsto; 2) che nelle altre righe ci siano esattamente N elementi.

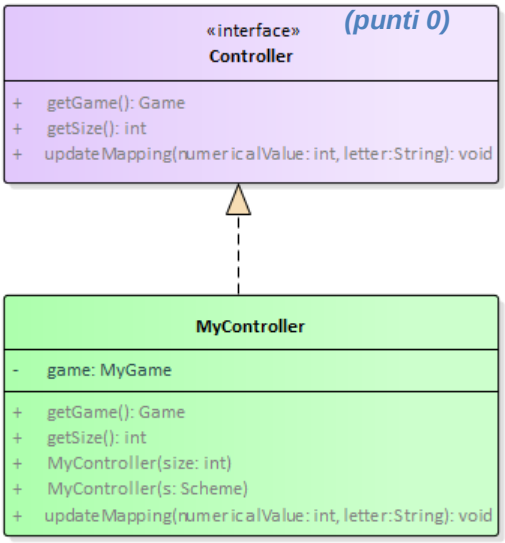
Parte 2)

[TEMPO STIMATO: 20-30 minuti] **(punti: 8)**

Controller (crosswords.controller)

(punti 0)

Il Controller (fornito) è organizzato secondo il diagramma UML in figura.



SEMANTICA:

a) L'interfaccia ***Controller*** (fornita) dichiara i metodi *getSize, getGame* e *updateMapping*, dall'ovvio significato.

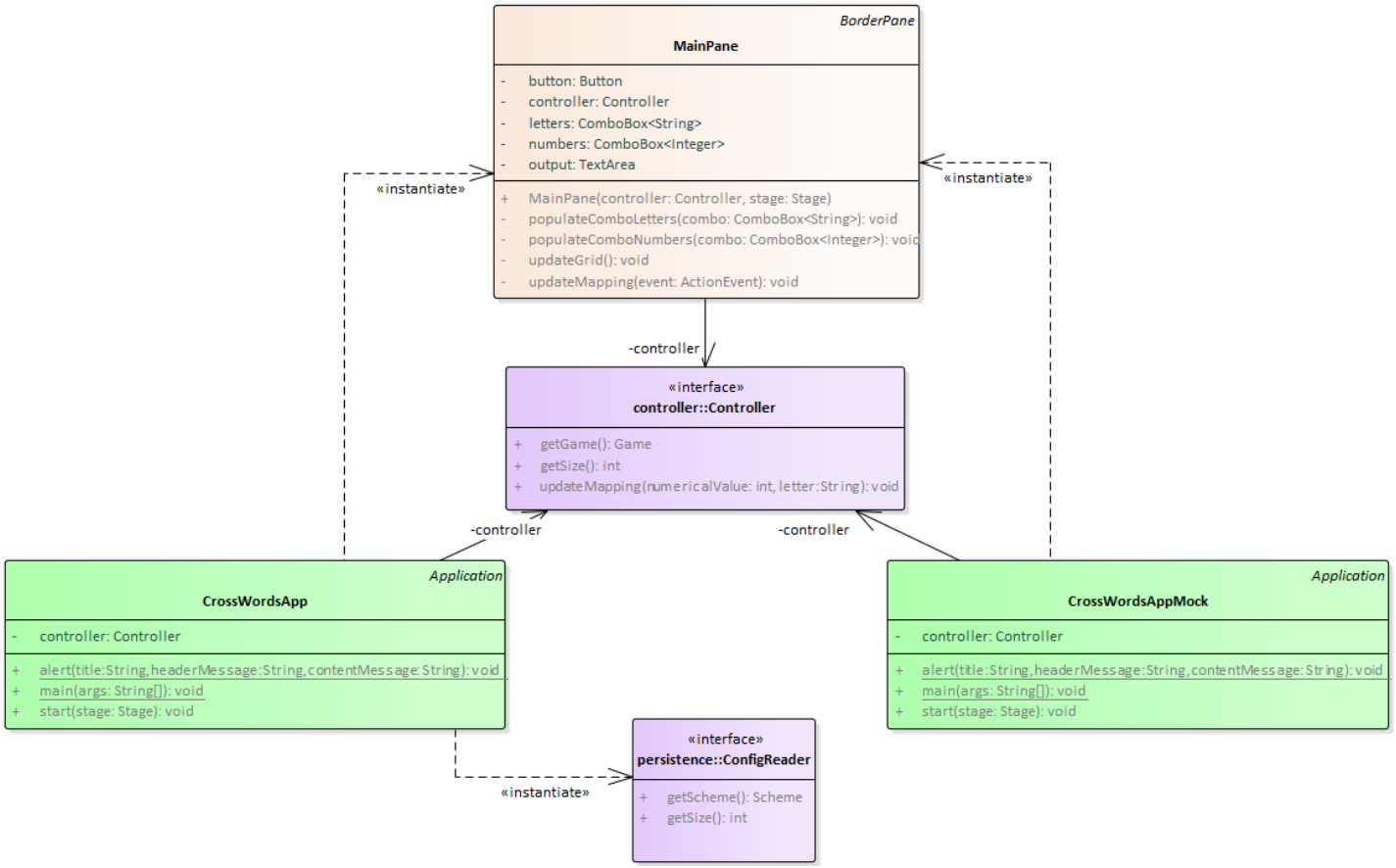
b) La classe ***MyController*** (fornita) implementa tale interfaccia, fornendo:

- il costruttore che, dato lo ***Scheme***, costruisce e gestisce il ***Game*** (NB: è presente anche un costruttore ausiliario utile a fini di test)
- implementa i tre metodi delegando il lavoro al retrostante ***Game***.

Interfaccia utente (crosswords.ui)

(punti 8)

L'interfaccia utente è illustrata nelle figure seguenti e segue il modello sotto illustrato:



La classe ***CrossWordsApp*** (fornita) costituisce l'applicazione JavaFX che si occupa di aprire i file, creare il controller e incorporare il ***MainPane***. Per consentire di collaudare la GUI anche in assenza / in caso di malfunzionamento della parte di persistenza, è possibile avviare l'applicazione mediante la classe ***CrossWordsAppMock***.

Entrambe le classi contengono anche il **metodo statico ausiliario *alert***, utile per mostrare avvisi all'utente.

Il MainPane è fornito *parzialmente realizzato*: è presente la parte strutturale, mentre manca la parte di popolamento combo e gestione degli eventi.

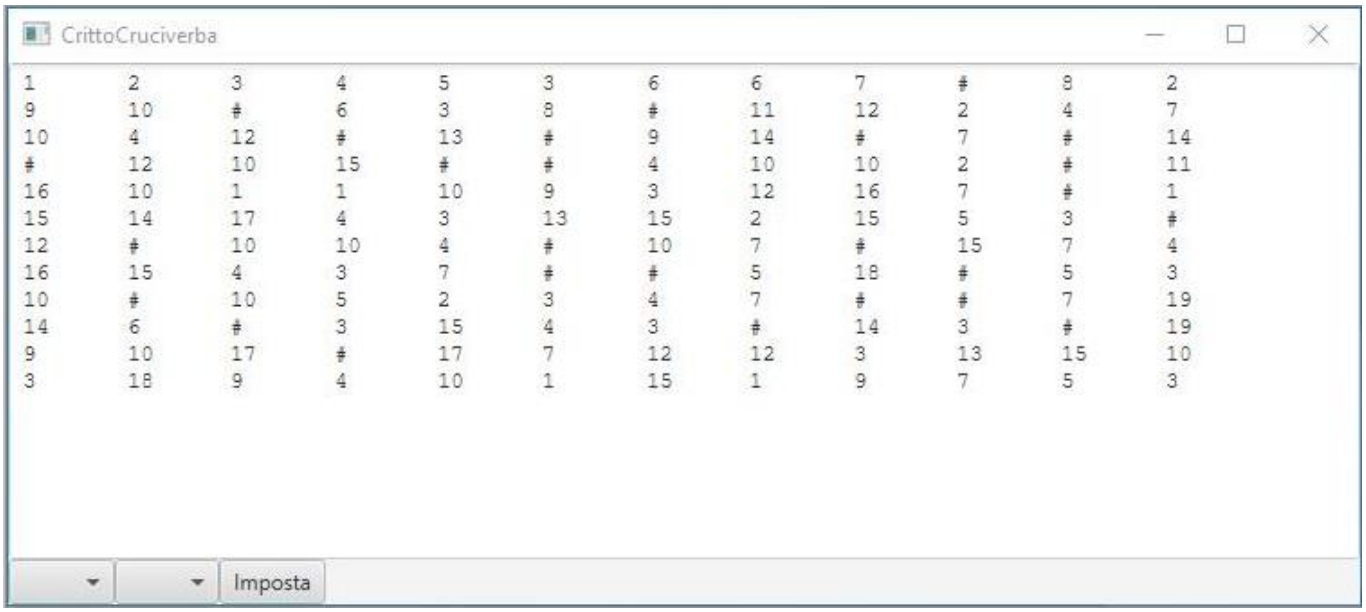
La classe ***MainPane*** (**da completare**) estende ***BorderPane*** e prevede:

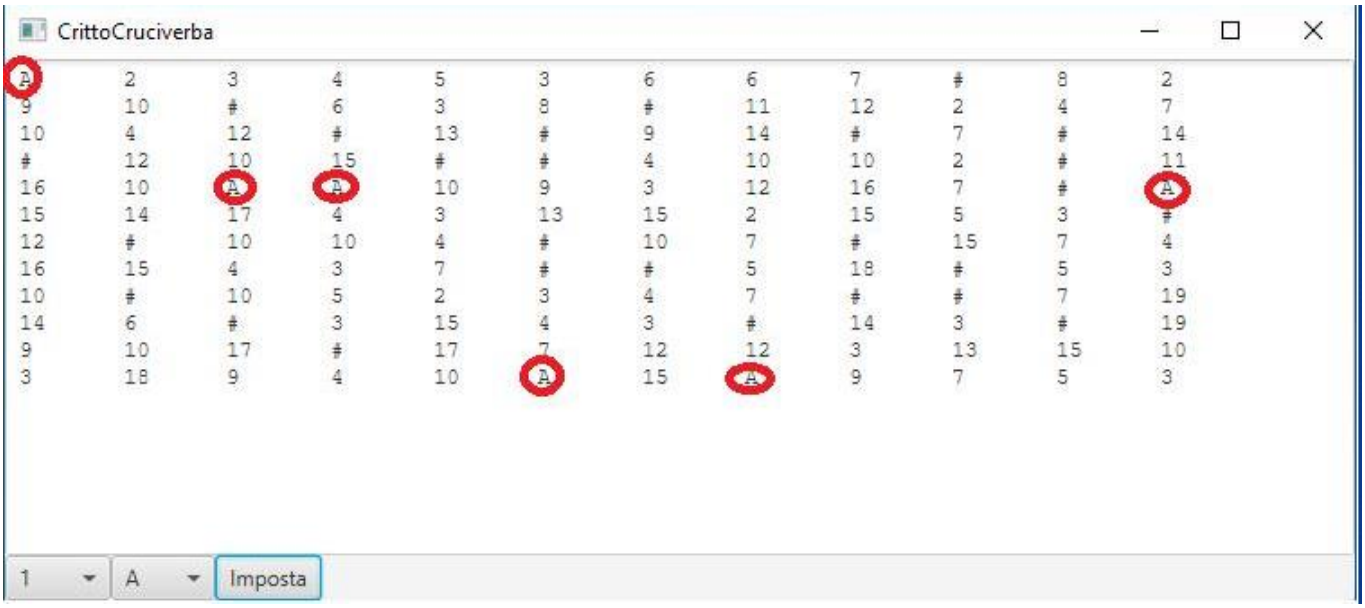
1) in alto, una ***TextArea*** che lo stato attuale del cruciverba
2) in basso, due ***ComboBox*** da popolare rispettivamente con i numeri da 1 a 26 (la prima) e le lettere da A a Z (la seconda)
3) a lato delle combo, un ***Button*** che imposta il mapping selezionato nelle combo.

La **parte da completare** comprende l'uso e/o l'implementazione dei seguenti metodi:

1) *populateComboNumbers* e *populateComboLetters* che popolano rispettivamente la combo dei numeri e la combo delle lettere
2) *updateMapping,* da chiamare in risposta all'evento di pressione del ***Button***
3) *updateGrid,* usato sia in *updateMapping* (questo dovete farlo voi) sia all'inizio per configurare lo stato iniziale (e questo lo abbiamo già fatto noi)

Inizialmente (Fig. 1), l'area mostra lo schema statico. Poi, ogni volta che l'utente sceglie un numero, una lettera e la imposta, la relativa associazione viene registrata e la visualizzazione si aggiorna di conseguenza (Fig. 2).





Cose da ricordare

- salva costantemente il tuo lavoro: l'informatica a volte può essere "subdolamente ostile"..
- in particolare: se ora compila e stai per fare modifiche, salva la versione attuale (non si sa mai)

Checklist di consegna

- Hai fatto un **JAR eseguibile**, che contenga cioè l'indicazione del main?
- Hai controllato che **si compili** e **ci sia tutto**? [NB: non includere il PDF del testo]
- Hai **rinominato** IL PROGETTO, lo ZIP e il JAR esattamente come richiesto?
- Hai **chiamato** la cartella del progetto esattamente come richiesto?
- Hai fatto un **unico file ZIP (NON .7z, rar o altri formati)** contenente l'intero progetto? In particolare, ti sei assicurato di aver incluso tutti i file .java (e non solo i .class)?
- **Hai consegnato DUE file distinti, ossia lo ZIP col progetto e il JAR eseguibile?**
- Su EOL, hai **premuto** il tasto "CONFERMA" per inviare il tuo elaborato?