ESAME DI FONDAMENTI DI INFORMATICA T-2 del 7/7/2020

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 ***Sudoku***. L'obiettivo non è giocare "contro" il computer o "farselo risolvere" dal computer, ma semplicemente predisporre un supporto informatico che consenta al solutore di inserire i vari numeri senza dover ricorrere a carta e matita.

DESCRIZIONE DEL DOMINIO DEL PROBLEMA

Sudoku è un gioco di logica e abilità costituito da una *griglia* di 9×9 celle, ciascuna delle quali può contenere un numero da 1 a 9 o essere vuota; la griglia è suddivisa in:

- 9 righe orizzontali
- 9 colonne verticali
- 9 "sotto-griglie" di 3×3 celle contigue.

Queste sotto-griglie sono delimitate da bordi in neretto e chiamate *regioni*. Le griglie proposte al giocatore hanno da 20 a 35 celle già pre-inizializzate con un numero.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Lo scopo del gioco è riempire le caselle bianche con i "giusti" valori da 1 a 9 in modo tale che in ogni riga, in ogni colonna e in ogni regione siano presenti tutte le cifre da 1 a 9, ma senza ripetizioni.

ESEMPIO (foto): nello schema iniziale in alto, si consideri la terza sotto-griglia (quella in alto a destra): risulta ovvio che il numero 5 vada collocato a sinistra del 6 già presente, in quanto è l'unica scelta possibile data la collocazione degli altri 5 nelle prime due righe in alto e nell'ultima colonna a destra.

Per curiosità, l'immagine a lato mostra il corrispondente schema risolto.

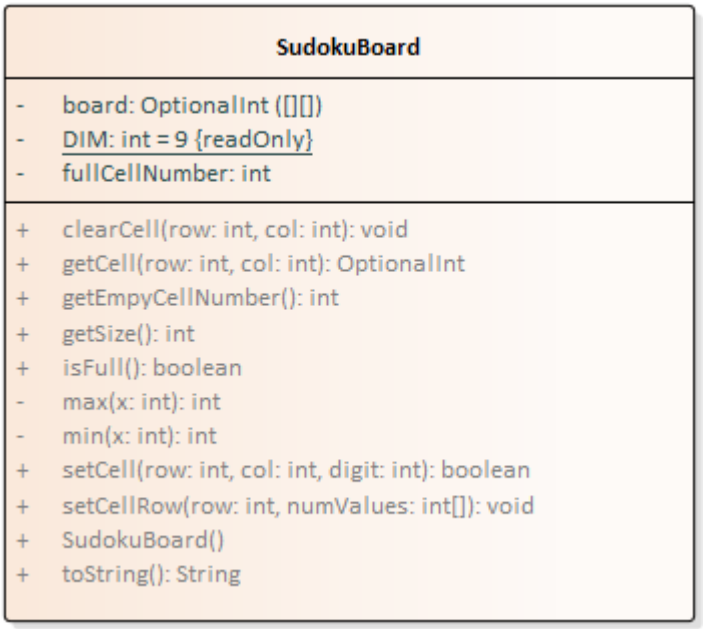
5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Lo schema di partenza 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 sudoku.model) [TEMPO STIMATO: 1h-1h10 minuti] *(punti: 12)*



SEMANTICA:

a) la classe ***SudokuBoard*** (**da realizzare**) rappresenta lo schema di gioco, una *scacchiera quadrata NxN* di ***OptionalInt***. La classe deve mettere a disposizione i seguenti metodi:

- la dimensione della griglia *N*, impostata a 9, deve essere memorizzata in un'apposita costante
- *costruttore* crea la scacchiera e la inizializza nel modo opportuno (corrispondente alla scacchiera vuota)
- *setCellRow* consente di caricare nello schema un'intera riga di valori. Per convenzione, nell'array che viene passato in ingresso al metodo le celle vuote sono indicate con il valore 0. Il metodo deve controllare accuratamente i parametri ricevuti (sia le dimensioni che i valori), lanciando ***IllegalArgumentException*** in caso di non conformità, con adeguata messaggistica.
- *getCell* restituisce un ***OptionalInt*** che rappresenta il contenuto della cella relativa agli indici ricevuti come argomento. Come sopra, il metodo deve controllare accuratamente i parametri ricevuti, lanciando ***IllegalArgumentException*** in caso di non conformità.
- *getSize* restituisce la dimensione *N* della scacchiera (numero di celle per ogni riga o colonna)
- *clearCell* "pulisce" il contenuto della cella relativa agli indici passati come argomento. Al solito, il metodo deve controllare i parametri ricevuti, lanciando ***IllegalArgumentException*** se necessario.
- *setCell* imposta la cella relativa alle coordinate ricevute in ingresso al valore ricevuto anch'esso in ingresso come terzo argomento, verificando preventivamente che tale inserimento rispetti le regole del SUDOKU: in caso positivo restituisce *true*, mentre se così non è non inserisce nulla e restituisce *false.* Anche qui, il metodo deve controllare i parametri ricevuti, lanciando ***IllegalArgumentException*** se necessario.
- *getEmptyCellNumber* restituisce il numero di caselle ancora vuote.
- *toString* restituisce una stringa con la struttura della scacchiera (in righe) intesa come sequenza dei numeri che etichettano le varie caselle, *utilizzando il carattere " " per indicare le caselle vuote.*

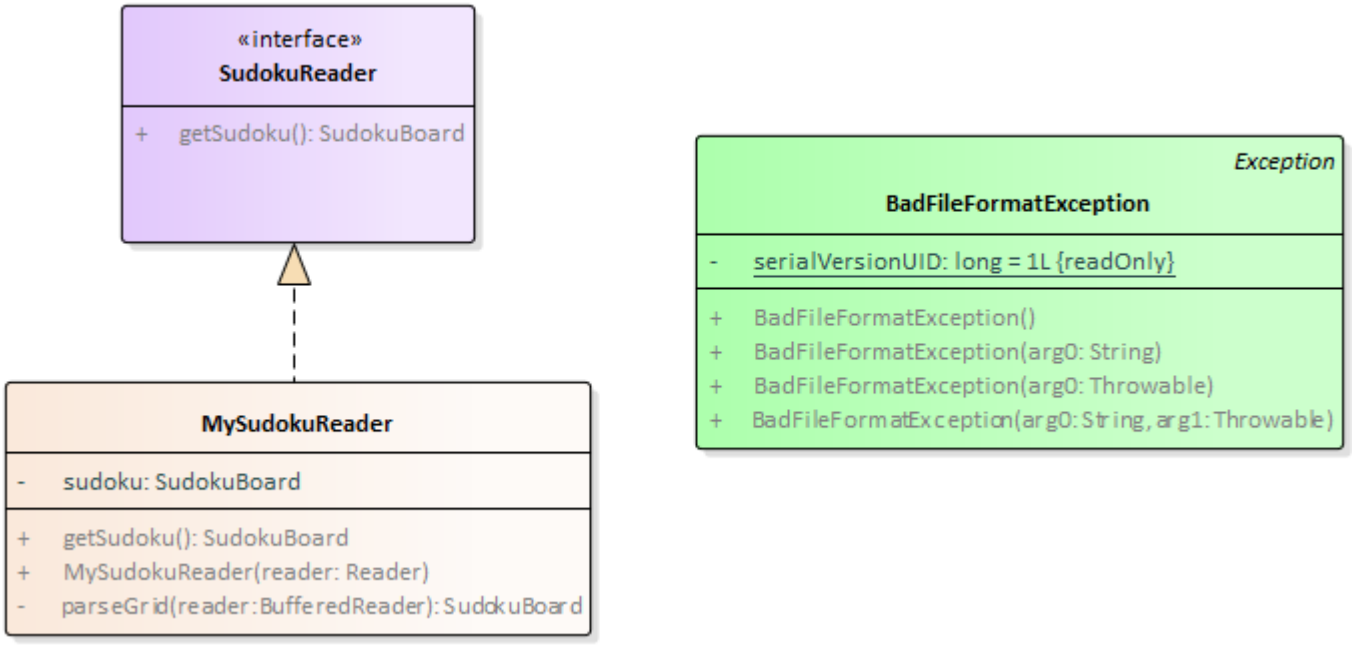
Persistenza (sudoku.persistence) **[TEMPO STIMATO: 30-40 minuti]** *(punti 10)*

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

- ogni riga nel file di testo descrive una riga della scacchiera del sudoku, che a sua volta conterrà i dati delle rispettive colonne (nel caso della scacchiera classica 9x9, ci saranno quindi 9 righe ognuna delle quali descriverà 9 colonne): è esplicitamente richiesto di non cablare soluzione sulla dimensione 9 del sudoku tradizionale
- in ogni riga i valori numerici sono separati fra loro da tabulazioni, mentre le celle vuote sono rappresentate dal carattere "#"

Esempio:

```
7  #  #  5  8  #  #  #  3
#  #  4  #  #  #  #  2  #
...
```



SEMANTICA:

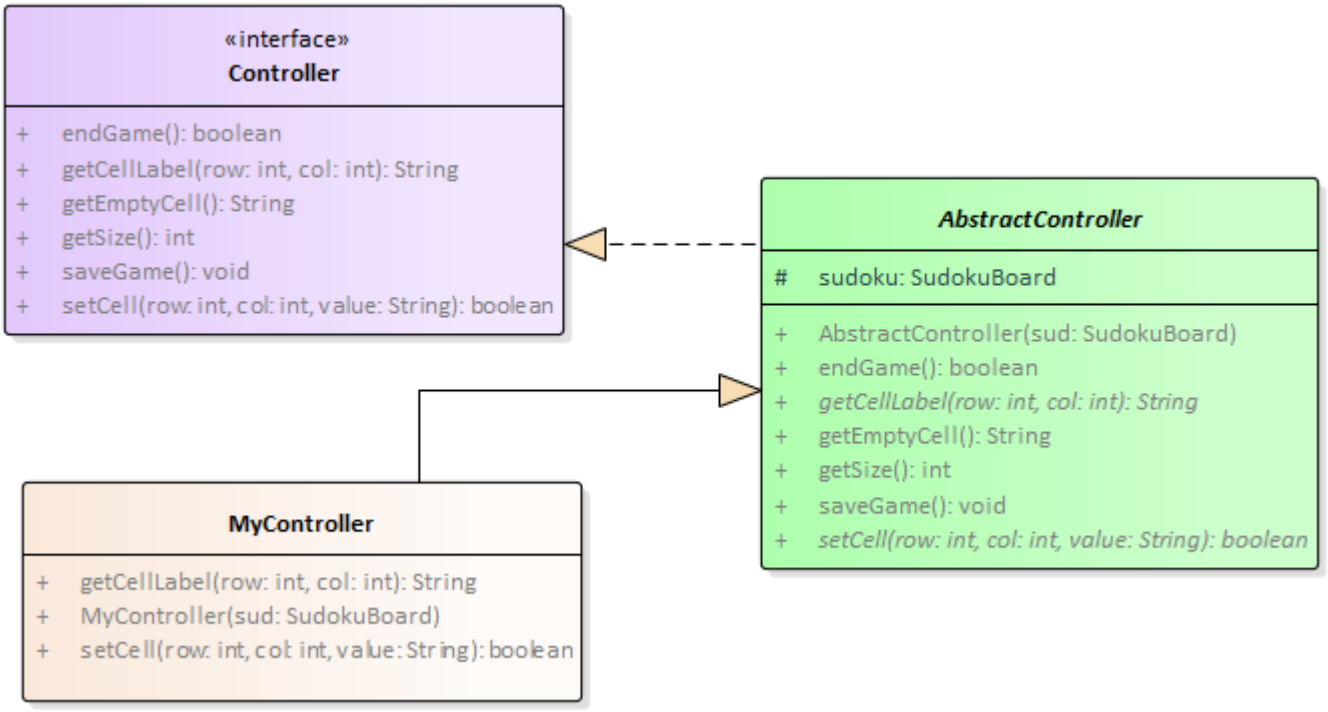
a) L'interfaccia ***SudokuReader*** (fornita) dichiara il metodo *getSudoku* che restituisce la scacchiera letta

b) La classe ***MySudokuReader*** (**da realizzare**) implementa ***SudokuReader***: il costruttore riceve un ***Reader*** già aperto ed effettua la lettura, costruendo e memorizzando internamente il ***SudokuBoard*** corrispondente. L'accessor specificato da ***SudokuReader*** restituisce tale ***SudokuBoard***. 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 ogni riga contenga esattamente 9 item e che ci siano esattamente 9 righe; 2) che i numeri presenti siano tutti compresi tra 1 e 9.

Parte 2 (punti: 8)

Controller (package sudoku.controllerI) [TEMPO STIMATO: 10-15 minuti] *(punti: 4)*

Il Controller è organizzato secondo il diagramma UML in figura.



SEMANTICA:

a) L'interfaccia ***Controller*** (fornita) dichiara i metodi

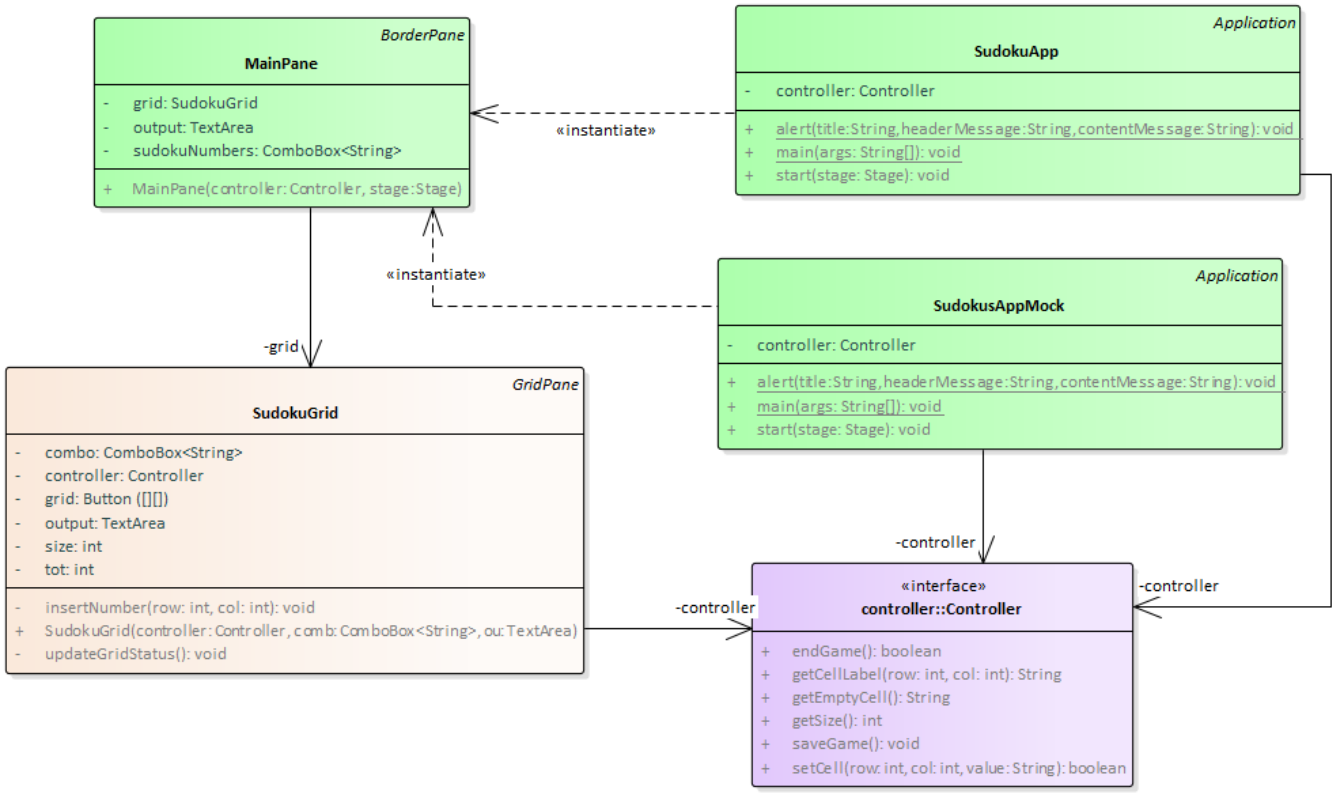
- *getSize* restituisce la dimensione *N* della scacchiera intesa come numero di celle per riga (o colonna)
- *getCellLabel* restituisce il contenuto della cella specificata in forma di stringa
- *setCell* imposta la cella relativa alle coordinate ricevute in ingresso al valore ricevuto anch'esso in ingresso come terzo parametro, richiamando gli opportuni metodi del model per effettuare l'operazione richiesta. Restituisce true se l'operazione è andata a buon fine, false altrimenti
- *endGame* restituisce true se la scacchiera è tutta piena
- *saveGame* salva su File di testo la soluzione ottenuta
- *getEmptyCell* restituisce il numero di celle ancora da riempire

b) `La classe` ***AbstractController*** (fornita) implementa l'interfaccia ***Controller*** e lascia astratti i due metodi *setCell* e *getCellLabel*

c) La classe ***MyController*** (**da realizzare**) estende la classe ***AbstractController***, fornendo l'implementazione ai due metodi astratti *setCell* e *getCellLabel* secondo quanto specificato nell'interfaccia ***Controller***

Interfaccia utente (package sudoku.ui) **[TEMPO STIMATO: 10-15 minuti]** *(punti: 4)*

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



La classe ***SudokuApp*** (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 ***SudokuAppMock***.

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

La classe ***MainPane*** (fornita) estende ***BorderPane*** e prevede:

1) nel lato sinistro, una ***Label***, una ***ComboBox*** che contiene i numeri da inserire nella scacchiera del Sudoku, una ***TextArea*** che indica quante caselle sono ancora da riempire nella scacchiera
2) nella parte centrale, una ***SudokuGrid***.

La classe ***SudokuGrid*** **è fornita parzialmente realizzata: è presente la parte strutturale, mentre manca la parte di gestione degli eventi**. Questa classe estende un ***GridPane*** e rappresenta una griglia di pulsanti.

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

1) *updateGridStatus,* usato sia all'inizio per configurare lo stato iniziale (e questo lo abbiamo già fatto noi) sia in *insertNumber* (questo lo dovete fare voi) inserisce la corretta label su ogni bottone e aggiorna la TextArea in base al numero di celle che sono ancora da riempire (Fig.1)
2) *Il metodo insertNumber* reagisce alla pressione dei bottoni che rappresentano la scacchiera e deve eseguire le seguenti azioni:
    - recupera il valore della ***ComboBox***
    - prova ad impostare tale valore nella cella
    - se non è possibile, segnala errore (Fig.2)
    - verifica se il gioco è terminato e nel caso provvede a salvare su file la scacchiera completa, mostrando contestualmente a video un messaggio di complimenti (Fig.3)

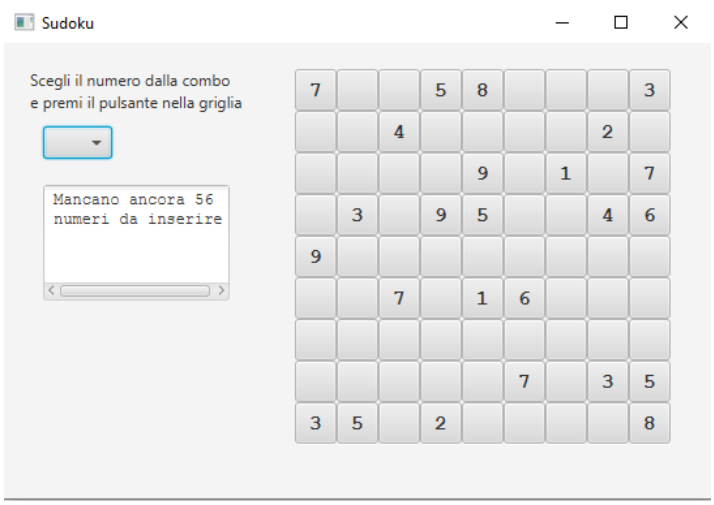


Fig.1

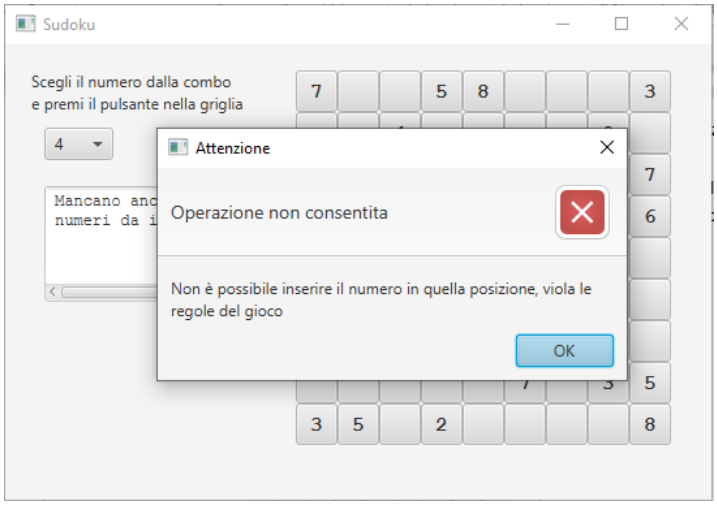


Fig.2

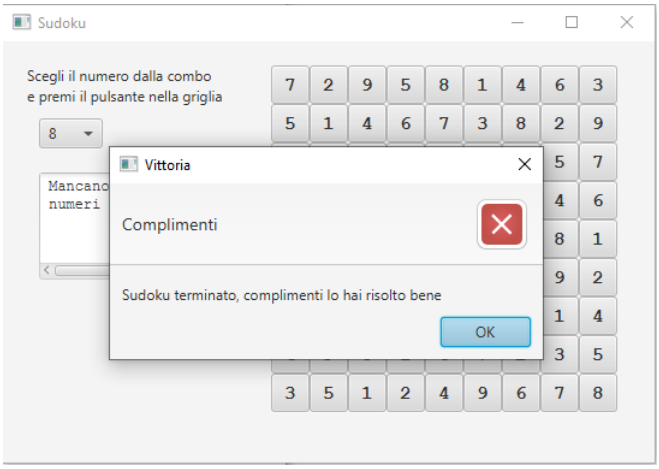


Fig.3

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?