POO – Lezione 7

Esercizio 1:
Si realizzi un'applicazione Java in grado di gestire i personaggi di un videogioco garantendo i seguenti requisiti e/o funzionalità:

Definire l'interfaccia **ComandiBase** che espone i seguenti metodi:

- boolean attacco(Player other)
- void potenziamento()

Definire l'interfaccia **ComandiAvanzati** che espone i seguenti metodi:

- boolean fusione(Player two) throws IllegalActionException
- Boolean attaccoAereo(Player other) throws IllegalActionException

Definire la classe **Player** che modella le caratteristiche comune di un personaggio in modo astratto; quali: i punti vita, il valore di attacco, il valore della difesa e il numero di uccisioni effettuate. Tale classe deve implementare le due interfacce definite in precedenza.

Definire la classe **Lottatore** che estende la classe Player ed implementa i metodi astratti. In particolare, si garantisca che:

- Ogni lottatore abbia i punti vita pari al doppio di un Player generico
- Ogni lottatore abbia il valore dell'attacco pari al quadruplo di un Player generico.
- Il metodo attacco sia invocato solo se entrambi i player sono ancora in vita e se l'attacco dell'attaccante è maggiore della difesa dell'attaccato.
- Il metodo attacco restituisce true se l'attacco è andato a buon file; false altrimenti.
- Il metodo attacco toglie all'avversario un numero di punti vita pari al doppio dell'attacco dell'attaccante.
- Il metodo attacco controlla se, dopo l'attacco, i punti vita dell'avversario sono minori o uguali a zero. In tal caso incrementa di uno il numero di uccisioni fatte dall'attaccante.
- Il metodo potenziamento può essere invocato solo se il player è ancora in vita e se ha fatto almeno un numero di uccisioni maggiore di 5.
- Il metodo potenziamento incrementa il valore della difesa del doppio e setta il numero di uccisioni a zero.

- Il metodo attaccoAereo non può essere invocato da un lottatore. Se accade lanciare opportunamente l'eccezione indicata.
- Il metodo fusione controlla se i player sono Lottatori in vita e unisce il secondo lottatore al primo. In particolare, i punti vita e l'attacco saranno pari alla somma dei punti vita e degli attacchi dei due Lottatori, mentre la difesa sarà uguale a quella del secondo Lottatore. Tale metodo, inoltre setta a null il secondo lottatore ed azzero il numero di uccisioni del primo player (il player risultante del processo di fusione).

Definire la classe **Mago** che estende la classe Player ed implementa i metodi astratti. In particolare, si garantisca che:

- Ci sia un livello di magia di default pari a 50.
- Non possano essere invocati i metodi fusione ed attaccoAereo; se fatto lanciare le eccezioni definite.
- Il metodo potenziamento effettui un potenziamento solo se il mago è ancora in vita e possiede un numero di uccisioni maggiore di 10.
- Il potenziamento dovrebbe incrementare il livello della magia di altri 50 punti, duplicare il valore dell'attacco, quadruplicare il valore della difesa e settare a zero il numero di uccisioni.
- Il metodo attacco sia invocato se entrambi i player sono ancora in vita e se l'attacco dell'attaccante è maggiore o uguale alla difesa dell'attaccato.
- In tal caso il danno inflitto all'avversario dovrà essere alla somma dell'attacco e del livello di magia dell'attaccante. Inoltre, incrementi il numero di uccisioni se il player attaccato sia stato sconfitto.
- Viceversa, sempre se i due player sono in vita, il metodo attacca toglie all'attaccante un numero di punti vita pari alla differenza tra la difesa dell'attaccante meno l'attacco dell'attaccante. In tal caso, verificare se l'attaccante viene sconfitto e, in tal caso, incrementare il numero di uccisioni del difensore.

Implementare una classe **Test** in cui creare una lista di Player contenente almeno 3 Lottatori ed almeno 3 Maghi.

Esercizio 2:
Migliorare l'applicativo sviluppato nell'esercizio precedente, in particolare la classe Test, implementando le opportune operazioni tramite espressioni lambda e stream:

- Stampa delle info di tutti i Player.
- Stampa delle info di tutti i Player con attacco e difesa >= 100 (o di una soglia K).
- Stampa delle info di tutti i Player con punti vita >= difesa.
- Stampa delle info di tutti e soli i Maghi.
- Stampa delle info di tutti e solo i Lottatori.
- Stampa dei soli valori d'attacco di tutti i Player.

Definire l'eccezione non controllata **StatLowerException**.

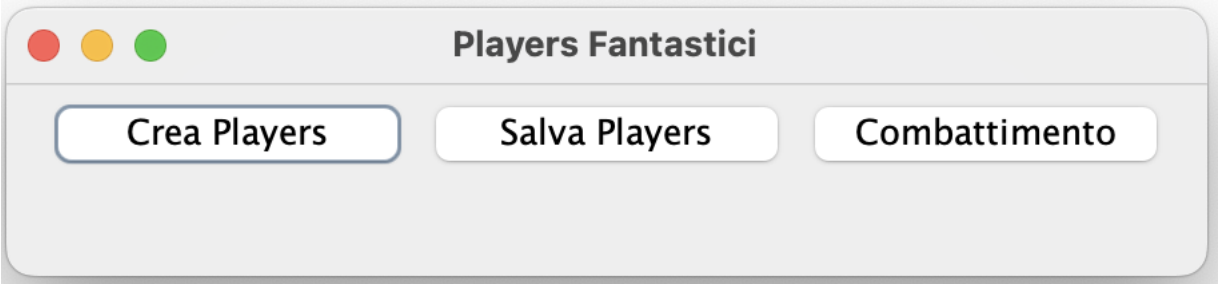
Implementare la classe **LottatoreVolante** che estende Lottatore tale che:

- Non possa essere creato un lottatore volante con attacco <= difesa. In tal caso gestire tutto tramite il lancio dell'eccezione definita in precedenza.
- Ridefinire il metodo fusione in modo tale che la fusione non sia più eseguibile; Lanciare un'eccezione opportuna.
- Ridefinire il metodo attaccoAereo in modo tale che sia eseguito quando i due player sono in vita e l'attacco dell'attaccante sia >= della difesa del difendente. In tal caso, diminuire i punti vita dell'avversario pari alla differenza tra i punti vita e l'attacco dell'attaccante. Come fatto in precedenza, incrementare opportunamente il numero di uccisioni.

Aggiungere alla lista già creata almeno 2 Lottatori volanti.

Rieseguire le operazioni precedenti di stampa e confrontare gli output.

Esercizio 3:
Implementare la seguente interfaccia grafica:



Inoltre, rispettare i seguenti requisiti:

- Gestire gli eventi associati ad ogni pulsante tramite il concetto di interfaccia funzionale.
- Il pulsante Crea Players deve creare una lista di Player (potete utilizzare le istruzioni e le add fatte nella classe Test).
- Il pulsante Salva Players salva su un file ogni player presente nella lista creata.
- Il pulsante Combattimento simula un combattimento tra due player selezionati in modo **random** e **rimossi** dalla lista dei player. (Vedi **Random** in java.util).
- Il combattimento deve durare al massimo 10 round (iterazioni). Ad ogni round un giocatore attacca e l'altro difende. Ad esempio, il Player uno attacca sempre ai round pari (%2==0) mentre il Player due attacca sempre ai round dispari.
- Ad ogni round il player attaccante tenta di attaccare il player difensore (segnalare a video l'esito) e tenta di potenziarsi.
- Alla fine di un combattimento (iterazione) il sistema stampa le statistiche aggiornate e controlla se uno dei player è stato sconfitto; proclamando eventualmente il vincitore dell'incontro.
- Il sistema inserisce il vincitore dell'incontro nella lista dei Player in modo tale da renderlo disponibile per un altro combattimento.
- Il sistema proclama il vincitore del torneo l'ultimo Player rimasto in vita (nella lista dei player) nel momento in cui si prova a far partire un combattimento.
- Esempio di output di un combattimento:

```
Attacca core.Lottatore

Attacco Riuscito


Lottatore [Player [puntiVita=200, attacco=600, difesa=50, numUccisioni=1]]
Mago [livMagia=50, Player [puntiVita=-1100, attacco=50, difesa=50, numUccisioni=0]]


Player: Lottatore [Player [puntiVita=200, attacco=600, difesa=50, numUccisioni=1]] vince!
```