Esercitazione 11

Prof. Tornese Gianluca
19/11/2024

Questa esercitazione mette insieme alcune delle vostre conoscenze OOP, in preparazione al progetto di sviluppo valutato.

Ci permette di realizzare un semplice gioco, dal nome FPGG (Fantasy Pascal Geeky Game).

Ci serviranno le seguenti classi ed interfacce:

- EventoNaturale
- Fulmine
- ElementoDelGrimorio (interfaccia)
- GameException
- Giocatore
- Grimorio
- Main
- Mostro
- Pozione
- Terremoto

Le classi sono qui di seguito dettagliate:

EventoNaturale

Questa classe è molto semplice, e ci permette per la prima volta di utilizzare il concetto di *abstract*. Questo costrutto è definito prima del nome della classe stessa, ed indica che la classe non potrà essere direttamente istanziata. Può invece comunque essere estesa. All'interno abbiamo solo bisogno di un attributo intero visibile alle classi figlie, chiamato *dannoInflitto*; non dobbiamo preoccuparci di inizializzarlo, essendo la classe *abstract* non è necessario.

Fulmine

Estende *EventoNaturale*, definisce un costruttore senza parametri (serve invocare *super* ?), che inizializza il *dannoInflitto* a 4.

ElementoDelGrimorio

Questa interfaccia è vuota, non offre metodi, e serve esclusivamente a dare una forma comune a oggetti di diversa natura, come vedremo più avanti. Ricordate il principio del "*what*" e non "*how*"?

GameException

Estende *Exception*, ha un attributo visibile alle classi figlie *message* di tipo Stringa, ricevuto ed inizializzato nel costruttore. Nell'invocare il metodo costruttore *super* passiamo il parametro *message*. Infine sovrascriviamo il metodo *toString*, restituendo esclusivamente *message*.

Terremoto

Come *Fulmine*, ma il danno inflitto è pari a 3.

Pozione

Questa classe implementa l'interfaccia *ElementoDelGrimorio*, e ha come attributi:

- un intero chiamato *valore*, visibile alle classi estese
- un'istanza di *Random* statica, creata contestualmente alla dichiarazione stessa; la variabile si chiama, con grande fantasia, *random*

Nel metodo costruttore, che non riceve parametri, inizializziamo l'attributo *valore* ad un valore casuale nell'intervallo [1, 10].

Come mai per *random* si è utilizzato il costrutto *static*?

Mostro

Molto simile a *Pozione*, al posto di *valore* definiamo *attacco*, che nel costruttore è inizializzato con valori casuali nell'intervallo [1, 15].

Grimorio

Da qui in avanti le cose si complicano leggermente…

Definiamo due attributi:

- *grimorio*, privato e di tipo *ArrayList<ElementoDelGrimorio>*
- *nomeDelGiocatore*, privato e di tipo Stringa

Nel costruttore:

- riceviamo ed inizializziamo *nomeDelGiocatore*, e poi istanziamo l'oggetto *grimorio*.
- a questo punto realizziamo un ciclo che per 15 volte crea un'istanza di *Pozione*, e la aggiunge a *grimorio* tramite la funzione *add*.
- allo stesso modo creiamo ed aggiungiamo 25 istanze di *Mostro*. Queste istanze avranno tutti valori interni diversi, grazie all'uso di *Random*.
- per ultimo invochiamo *Collections.shuffle* passando come parametro *grimorio*; cosa si vuole ottenere con questa operazione?

Ora definiamo un metodo *pesca*, senza parametri e con valore di ritorno *ElementoDelGrimorio*. All'interno controlliamo se *grimorio* è vuoto, con la funzione *isEmpty*,

e nel caso lanciamo una *GameException* con un messaggio che dica che *nomeDelGiocatore* ha terminato il grimorio. In caso contrario, restituiamo il primo elemento di *grimorio*, rimuovendolo con *remove* alla posizione 0.

Giocatore

Questa classe ha tre attributi, tutti privati:

- *nome*, di tipo Stringa
- *puntiVita*, di tipo int
- *grimorio*, di tipo *Grimorio*

Tutti gli attributi sono inizializzati nel metodo costruttore, che riceve solo il *nome* come parametro.

Iniziamo con la funzione *controllaPuntiVita*, privata, senza valore di ritorno e che non riceve nulla in ingresso. Se verifica che i *puntiVita* siano 0 o inferiori, lancia una *GameException* con un messaggio che dica che *nome* ha terminato i punti vita.

Definiamo ora la funzione *infliggiDanno*, pubblica, senza valore di ritorno e che prenda in ingresso un numero intero. Per prima cosa stampa a video un messaggio che mostri che il giocatore *nome* ha ricevuto alcuni punti ferita. A questo punto deve applicare il danno ricevuto, e verificare se questo causa la fine del gioco o meno, invocando *controllaPuntiVita*.

Per ultimo il metodo *giocaTurno*, pubblico, che restituisce un oggetto di tipo *Mostro* e che non prende nulla in ingresso. Per prima cosa invoca *controllaPuntiVita*, giusto per sicurezza. A questo punto invoca *pesca* sull'istanza *grimorio*, salvando il valore di ritorno in una variabile *e* di tipo *ElementoDelGrimorio*. Ora:

- se *e.getClass()* è equivalente a *Pozione*, effettuiamo un downcast a *Pozione*, otteniamo il *valore* della pozione, stampiamo a video che il giocatore *nome* recupera i punti e applichiamo l'incremento. Restituiamo *null* (il giocatore non ha pescato dal grimorio un mostro).
- altrimenti restituiamo il downcast di *e* a *Mostro*, pronti ad infliggere danni all'avversario

Possiamo ora passare al metodo *main*!

Metodo *main*

Procediamo come segue:

- stampiamo "Benvenuto nel gioco FPGG - Fantasy Pascal Geeky Game"
- definiamo e creiamo due istanze di *Giocatore*, con nomi a piacere
- definiamo e creiamo un'istanza di *Random* chiamata *randomEventiNaturali*
- inizializziamo un contatore *turno* a 0
- definiamo un ciclo infinito *while*, ed al suo interno:
    - incrementiamo e stampiamo a video a che turno di gioco siamo arrivati
    - invochiamo *Thread.sleep(250)* per riuscire a vedere nella console i turni in esecuzione

- definiamo due istanze di *Mostro*, *m1* e *m2*, inizializzate a *null*
- in un unico blocco di *try*:
  - invochiamo *giocaTurno* sul primo giocatore, memorizzando il risultato in *m1*
  - invochiamo *giocaTurno* sul secondo giocatore, memorizzando il risultato in *m2*
  - se *m1* è diverso da *null*, infliggiamo i danni di *m1* al secondo giocatore
  - se *m2* è diverso da *null*, infliggiamo i danni di *m2* al primo giocatore
  - generiamo un intero *evento* casualmente con *randomEventiNaturali* nell'intervallo [0, 10]
  - se *evento* è 0 o 1, creiamo un'istanza di *Fulmine* e infliggiamo il suo danno ad entrambi i giocatori
  - se *evento* è 2 o 3, creiamo un'istanza di *Terremoto* e infliggiamo il suo danno ad entrambi i giocatori
- catturiamo con il costrutto *catch* la nostra *GameException*, che stamperà a video che la partita è terminata, seguita dal *toString* dell'eccezione stessa; spezziamo il ciclo infinito con il costrutto *break*

The end!

Giocate alcuni turni, assicurandovi che tutto funzioni come atteso.

Questa esercitazione ha messo insieme i seguenti concetti, sui quali ragionare:

- access modifiers
- metodo toString
- keywords super ed extends
- keyword static
- exception handling
- interfacce
- upcast e downcast (impliciti ed espliciti)

Sapete ora modificare i sorgenti, per rendere il gioco più complesso e interattivo?