Esercizio 21

Realizzare utilizzando le tecniche proprie della OOP le classi descritte sotto, che implementeranno la struttura di un generico gioco di ruolo. Stabilire in autonomia, ove non espressamente indicato il tipo, tipo di ritorno o il modificatore di visibilità opportuno per gli attributi, le proprietà o i metodi.

Regole base:

L'attributo forza viene usato per determinare l'attacco, ovverosia, un numero random compreso tra 0 e il valore della forza. Ogni attacco prevede una parata (o difesa) da parte della creatura attaccata.

L'attributo destrezza viene usato per determinare i danni subiti dall'attacco, per cui:

- fase d'attacco: danno = numero random tra 0 e valore attacco
- fase parata: se destrezza < danno la creatura subisce una diminuzione dei punti salute pari a salute-danno

Esercizio 1

Creare la classe <u>astratta</u> Creatura che rappresenta una generica creatura.

<u>Attributi ed eventualmente Properties:</u>

- nome: stringa
- forza: intero
- destrezza: intero
- salute: intero
- fato: oggetto Random(), campo statico

<u>Metodi:</u>

- Creatura(...): costruttore di default che inizializza i campi membro
- attacca(Creatura c): metodo astratto che prende come parametro un oggetto di tipo Creatura. All'interno del metodo viene richiamato il metodo para() della creatura attaccata
- para(int danno): metodo astratto che prende in ingresso un intero rappresentante il danno generato dall'attacco
- diminuisciSaluteDi(int dannoEffettivo)
- isVivo(): vero se salute>0
- isSvenuto(): vero se salute=0
- isMorto(): vero se salute<0
- toString(): override del toString() della classe Object che restituisce lo stato interno dell'oggetto.

Esercizio 2

Derivare dalla classe Creatura la classe Goblin, che rappresenta uno dei tanti mostri del gioco.
Creare un costruttore che inizializzi con dei valori di default gli attributi della classe tenendo conto che il goblin non ha attributi nuovi rispetto alla classe derivata e non ha armi né armature e non è molto forte.
Implementare i due metodi astratti della classe base e modificare il toString() in modo tale da aggiungere la stringa "goblin" prima della descrizione dell'oggetto.

Esercizio 3

Creare una classe consumer (console) per testare la creazione di due oggetti goblin e farli combattere tra loro fino alla morte. Ad esempio

```
while (goblin1.isvivo() && goblin2.isvivo())
            {
                goblin1.Attacca(goblin2);
                goblin2.Attacca(goblin1);
            }
```

Esercizio 4

Creare la classe astratta Arma con i seguenti attributi e relative proprietà:

- danno:int, valore che andrà ad aggiungersi alla forza, se l'arma è integra
- integrità: int, valore che determina l'efficienza dell'arma, ad ogni attacco integrità decrementa di 1
- costo: double, valore dell'arma

e un costruttore di default che inizializzi gli attributi a valori opportuni

Derivare dalla classe Arma la classe Spada che è rappresenta una spada, con danno pari a 3, costo pari a 100 monete e integrità pari a 500.

Creare la classe astratta Armatura con i seguenti attributi e relative proprietà:

- protezione: int, valore che andrà ad aggiungersi alla destrezza, se l'armatura è integra
- integrità: int, valore che determina l'efficienza dell'armatura, ad ogni parata integrità decrementa di 1
- costo: double, valore dell'armatura

e un costruttore di default che inizializzi gli attributi a valori opportuni

Derivare dalla classe Armatura la classe ArmaturaPiastre che rappresenta una armatura di piastre, con un livello di protezione pari a 3 , un costo di 1000 monete e un livello di integrità pari a 1000.

Esercizio 5

Derivare dalla classe Creatura la classe Guerriero. Il guerriero, per sua natura, può possedere un'arma e una armatura, inoltre estende la classe base con in seguenti nuovi attributi:

- livello: intero, determina il livello del guerriero (per scopi futuri)
- modForza: intero, determina punti aggiuntivi di forza derivanti dalle armi
- modDestrezza: intero, determina punti aggiuntivi di destrezza derivanti dalle armature
- arma: di tipo Arma, null se non ha armi
- armatura: di tipo Armatura, null se non ha armature

Creare un costruttore che inizializzi con dei valori di default gli attributi della classe tenendo conto che il guerriero ha nuovi attributi rispetto alla classe derivata, ha armi e armature che influiscono sui metodi attacca e para ed è, generalmente, più forte del goblin.
Implementare i due metodi astratti della classe base e modificare il toString() in modo tale da aggiungere la stringa "guerriero" prima della descrizione dell'oggetto.

Provare ora nella classe consumer a far combattere fino alla morte il goblin e il guerriero

Esercizio 6

Creare nella classe consumer due eserciti composti sia da goblin che da guerrieri (un array di Creature) e far combattere i due eserciti fino alla morte. Per semplicità la creatura i-esima del primo array combatterà fino alla morte solo con la creatura i-esima del secondo array.
Quando ogni creatura i-esima nei rispettivi eserciti avrà sconfitto il suo omologo calcolare l'esercito vincitore basandosi sul numero di sopravvissuti.

Regole aggiuntive incantesimi:

Nell'universo di gioco esistono due tipologie di incantesimi, incantesimi di attacco (causa danni alla creatura bersaglio), incantesimi di difesa (aumentano temporaneamente la destrezza). Soltanto i maghi possono lanciare incantesimi, infatti solo tale classe possiede una scorta di energia magica (mana) che permette ai maghi di lanciare incantesimi con efficacia. Esaurito il mana un mago non potrà più lanciare incantesimi se prima non recupera l'energia magica (risposandosi o bevendo delle pozioni). Inoltre per lanciare incantesimi potenti bisogna essere maghi esperti (insomma maghi di certo livello☺). Gli incantesimi di difesa hanno una durata limitata nel tempo, di solito si esauriscono dopo 3 parate.

Esercizio 7

Creare la classe astratta Incantesimo con i seguenti attributi e relative proprietà:

- nome:string
- livello richiesto: int, determina il grado di competenze richieste al mago
- mana richiesto: int, determina l'energia magica spesa per lanciare l'incantesimo

Creare inoltre

- un costruttore di default che inizializzi gli attributi a valori opportuni
- un metodo privato isLanciabile() che accettando, in ingresso un oggetto di tipo "mago", determini se il mago sia effettivamente in grado di lanciare l'incantesimo
- un metodo astratto lancia() che accetta in ingresso 2 oggetti, il mago che lancia l'incantesimo e la creatura bersaglio dell'incantesimo.

Derivare dalla classe astratta Incantesimo la classe PallaDiFuoco che rappresenta un incantesimo d'attacco che crea una palla di fuoco che causa un danno X alla salute della creatura bersaglio. Ha l'attributo aggiuntivo danno di tipo intero, il livello richiesto è Y e il mana richiesto è Z.

Derivare dalla classe astratta Incantesimo la classe ScudoMagico che rappresenta un incantesimo di difesa che crea uno scudo protettivo che aumenta temporaneamente la destrezza della creatura bersaglio. Ha l'attributo aggiuntivo protezione di tipo intero, il livello richiesto è Y e il mana richiesto è Z.

Esercizio 8

Derivare dalla classe Guerriero la classe MagoGuerriero che estende la classe base con due nuovi attributi:

- mana, di tipo intero (rappresentante l'energia magica posseduta dal mago)
- effettiIncantesimiDifesa: una lista di interi, ogni elemento della lista rappresenta un aumento della destrezza derivante da uno o più incantesimi di difesa.

Creare N nuovi metodi per il lancio degli incantesimi, dove N rappresenta il numero di incantesimi presenti nell'universo di gioco. Per semplicità puoi fare in modo che il mago lanci la palla di fuoco direttamente nel metodo ridefinito attacco() e fare altrettanto col metodo para().
Di norma un mago può portare armi e armatura ma solo se sono leggere, inoltre i suoi attributi di forza e destrezza sono inferiori rispetto a quelli della classe Guerriero

Provare ora a rifare l'esercizio 5 inserendo oggetti di tipo magoGuerriero.