PROVA di LAB. classe 3IX as 2022/2023

Ereditarietà e polimorfismo: EROI vs LICANTROPI vs SUPEREROI

Viene fornita la cartella di lavoro **EREDITARIETA_3IX_STUDENTE1_STUDENTE2** che va salvata nel disco Z e rinominata opportunamente. Solo uno studente deve consegnare la prova per entrambi.

PREMESSA

Leggi attentamente il testo per comprendere pienamente l'esercizio. Se necessario **rileggilo più volte**!

Prima di scrivere codice java sarà necessario produrre un diagramma UML completo e corretto (usa jetUML e salva il diagramma in formato immagine). I nomi delle variabili di esemplare e dei metodi che andrai a scrivere nel diagramma dovranno coincidere con quelli dell'implementazione.

NB: Tutte le variabili di esemplare devono essere dichiarate **private**
NB: Per ogni classe devi prevedere i metodi getter, i setter e il toString() se non ereditati!
NB: Ricorda, in una classe astratta è buona regola implementare sempre i getter, i setter e il toString()

TESTO DELLA PROVA

Disegna un diagramma UML e scrivi codice java che descrive lo scenario che segue:

SCENARIO

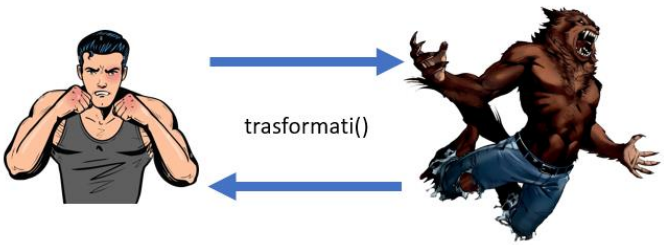
In un ipotetico gioco, un **Combattente** è un'entità astratta che ha un nome e uno stress (inteso come valore numerico intero, inizialmente a zero, che indica una sorta di affaticamento accumulato dopo ipotetici combattimenti con gli avversari). Se lo stress arriva a un valore superiore o uguale a 100, il combattente muore!

Un combattente è in grado di svolgere le seguenti azioni:

1. Si trasforma in qualcos'altro (*metodo astratto*)
2. E' in grado di stabilire di essere vivo o morto
3. Se vivo, può attaccare un avversario (che, automaticamente, si deve difendere). In questo caso accumula 5 punti stress
   Esempio: l'azione *combattente.attacca(avversario)* -> implica sempre *avversario.difenditi()*
4. Se vivo, può difendersi dall'attacco! In questo caso accumula 10 punti stress
5. Se vivo, può riposarsi! In questo caso azzera completamente lo stress accumulato nei combattimenti precedenti.

Un **Licantropo** è un combattente che, essendo un mutaforma per natura, possiede anche un nome da "lupo mannaro" ed è sempre possibile stabilire in che stato si trova (per intenderci … se è in modalità umano o in modalità lupo).

1. Nasce (viene istanziato) come "lupo mannaro", se c'è la luna piena, al contrario nasce come umano.
2. Il metodo *getNome()* deve restituire il giusto nome a seconda che si trovi nello status di umano o nello status di lupo.
3. In ogni momento, può trasformarsi da umano a lupo (e viceversa).
   Esempio figurativo



Un Licantropo umano Un licantropo "lupo mannaro"

4. In fase difensiva, accumula:
    a. 5 punti stress se è un lupo mannaro
    b. 10 punti stress se è umano
5. Il toString() deve restituire una stringa del tipo:
    a. **"Nome: Jacob – stress accumulato: 50 (Licantropo)"** nell'ipotesi che *Jacob* sia il nome da umano
    b. **"Nome: Bestia – stress accumulato: 50 (Licantropo)"** nell'ipotesi che *Bestia* sia il nome da lupo mannaro

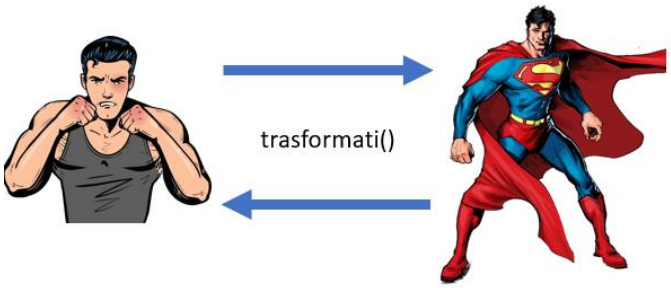
Un **Eroe** è un combattente che non ha alcun potere di trasformazione

1. Può attaccare solo Licantropi, in caso contrario viene visualizzato a video un messaggio di diniego
2. In fase di attacco accumula 15 punti stress

Un **SuperEroe** è un eroe che possiede un superpotere (inteso come una stringa che può assumere il valore "ATTIVATO" o "DISATTIVATO" a seconda dei casi).

1. Nasce (viene istanziato) inizialmente senza superpotere ma lo può acquisire (o perdere) per effetto di un processo di trasformazione.

    Esempio figurativo



Un SuperEroe senza superpotere Un SuperEroe con il superpotere

2. Non può attaccare altri SuperEroi, in caso contrario viene visualizzato a video un messaggio di diniego
3. se **NON HA** il superpotere,
    a. in fase di attacco o di difesa, si comporta esattamente come un Eroe
4. Se **HA** il superpotere,
    a. in fase di difesa non accumula stress, in fase di attacco annienta l'avversario.

FINE SCENARIO

Scrivi il codice java per risolvere l'esercizio sfruttando il più possibile l'ereditarietà e il polimorfismo utilizzando, se necessario, le parole chiave:

a) super
b) instanceof
c) l'invocazione del metodo nomeOggetto.getClass().getName() che restituisce il nome della classe di appartenenza di un oggetto

Scrivi 3 classi tester: **TesterLicantropo**, **TesterEroe**, **TesterSuperEroe**. In ognuna dovrai creare un combattente (Licantropo, Eroe o SuperEroe a seconda della classe tester che stai implementando) che attaccherà 3 avversari combattenti che andrai a inserire in un ArrayList: un Licantropo di nome "Jacob/Bestia", un Eroe di nome "Mario Bros" e un Supereroe di nome "Superman".
Per ognuna testa tutte le possibili combinazioni.

VALUTAZIONE

Sarà condizione necessaria, ma non sufficiente, al fine di ottenere almeno voto 6, che l'intero progetto sia compilabile in presenza di un congruo numero di righe di codice funzionante.

Il bad-code (che provoca il fallimento della compilazione) oscurato da commenti non sarà preso in considerazione.

Per aspirare alla valutazione massima:

1. Consegnare il diagramma UML richiesto
2. Il progetto deve andare in esecuzione;
3. Tutte le consegne dovranno essere rispettate;
4. Il codice sorgente deve essere di qualità (ricorda che stai svolgendo una prova sull'ereditarietà)

Sarà condizione necessaria, ma non sufficiente, al fine di ottenere almeno voto 6, che l'intero progetto sia compilabile in presenza di un congruo numero di righe di codice funzionante.

Il codice non funzionante, oscurato da commenti, non sarà preso in considerazione.

Per aspirare alla valutazione massima:

- il progetto deve andare in esecuzione;
- tutte le consegne dovranno essere rispettate;
- il codice sorgente deve essere di qualità (evitare le ridondanze, riutilizzare correttamente il codice già scritto e ricordare che, nell'ereditarietà, i metodi possono essere sovrascritti ed possibile sfruttare al meglio il polimorfismo).