ESERCIZI IN JAVA

1) Scrivere un programma Java nel cui main viene svolto il seguente compito
2) Calcolare la media di N voti inseriti da tastiera
3) Dato come input il numero di secondi, convertire il tempo in ore, minuti e secondi
4) Una scuola è composta da N classi. Per ogni classe, il numero di studenti è generato in modo casuale (tra 15 e 30). Calcolare quanti frequentano la scuola e in media quanti studenti ci sono per classe, il numero minimo e il numero massimo. Dire se c'è almeno una classe con tutti assenti.
5) Scrivere un programma che legge da tastiera un intero e stampa un messaggio a video secondo la seguente tabella:
    1. *Valore* *Messaggio*
    2. v >= 700 Montagna
    3. 700 > v >= 300 Collina
    4. 300 > v >= 0 Pianura
6) Scrivere un programma java che legga da input una stringa rappresentante il nome di un mese dell'anno, e stampi un messaggio indicante il numero di giorni di quel mese [si considera sempre un anno come non bisestile].
7) Data una sequenza di stringhe con il seguente formato *cognome-nome-eta-sesso*, visualizzare per ogni stringa il cognome in maiuscolo e il nome con l'iniziale maiuscola. Inoltre calcolare l'età media delle donne e degli uomini. (Usare i metodi della classe String: toUpperCase, indexOf, substring, concat,…)

Oggetti in Java

Dei seguenti esercizi rappresenta il diagramma delle classi in UML e implementa la classe e un main di esempio in Java

8) Dichiarare una classe di nome “temperatura” che ha come attributo la temperatura in gradi centigradi (gradiC e i metodi getGradiC e setGradiC). Definire un costruttore per inizializzare la temperatura, e i metodi (get/setGradiF) per convertire la temperatura da gradi centigradi a Fahrenheit, sapendo che: F = 32 + (9 * C / 5)
9) Dichiarare una classe di nome “velocita” che ha come attributo la velocità in km/h (velKmH). Possiede solo il costruttore senza parametri che inizializza la velocità a 0. Definire i 4 metodi get e set che restituiscono/impostano la velocità in Km/h o in m/s.
10) La classe Cerchio deve avere due costruttori: uno senza parametri che imposta il raggio a 0 e uno con un parametro corrispondente al raggio che si vuole impostare. Deve fornire oltre ai metodi get/set opportuni anche i metodi getArea() e getCirconferenza(). Creare la classe Test che prova a creare 2 cerchi, uno con il costruttore senza parametri e poi usa il metodo setRaggio() e l'altro con il costruttore con un parametro. Quindi visualizzare la loro area e la loro circonferenza.
11) La classe rettangolo deve permettere di calcolare Area e Perimetro e di dire se è un quadrato. Possiede due costruttori uno senza parametri e uno con base e altezza passati come argomenti.
12) Date le classi Cerchio e Rettangolo degli esercizi precedenti costruire un main in cui dati un cerchio di raggio r e un rettangolo b x h dire se sono equivalenti (hanno la stessa area)
13) La classe TELECOMANDO deve essere in grado di: accendere/spegnere la televisione, cambiare il canale (su di uno o giù di uno) o impostare un canale (se esiste se no resta invariato), abbassare e alzare il volume o mettere a muto. Usare gli attributi privati: canale, acceso, volume, VOLUME_MAX, CANALE_MAX
14) Scrivere la classe Motorino con i seguenti attributi: colore, velocità in km/h, tipo (marca + modello), antifurto (boolean che esprime se l'antifurto è inserito oppure no), velocitàMax. Il costruttore ha come parametri un colore, un tipo e una velocità massima. Scrivere il metodo getVelocità che restituisce la velocità del motorino, il metodo accelera(diX) che incrementa diX la velocità se non è inserito l'antifurto, se no non fa nulla e il metodo inserisciAntifurto oltre ai metodi get/set che ritieni opportuni
15) Dichiarare una classe di nome “Persone” che ha come attributi un vettore di tipo string (nomi) e un attributo di tipo char (vocale). Definire un costruttore per inizializzare la vocale, e 2 metodi, il primo (add) per aggiungere un nome, il secondo (visualizza) per visualizzare tutti i nomi che iniziano con quella vocale.
16) Scrivere una classe Java *Nominativo* che possiede gli attributi cognome, nome, giorno mese e anno di nascita, sesso e oltre ai soliti set/get e toString, anche i metodi:
    - *iniziali*() che restituisce: le prime tre lettere del nome in minuscolo concatenate con le prime tre lettere del cognome in maiuscolo. Utilizzare il metodo concat ed il metodo substring. Se non ci sono 3 lettere aggiungere delle X finali.

- *inizialiCF*() che restituisce i primi 11 caratteri del Codice Fiscale in maiuscolo (cercare su internet le regole).
- *eOmocodice* (*Nominativo altroNom*) che restituisce true se l'oggetto su cui è invocato genera lo stesso CF dell'oggetto *altroNom* passato per parametro

17) Utilizzate un array unidimensionale per risolvere questo problema: creare 20 numeri casuali compresi tra 10 e 100. Per ogni numero creato, inseritelo solo se esso non è già stato immesso in precedenza.
18) Creare la classe *Data* con gli attributi numerici *giorno, mese e anno* e l'attributo *numeroGiorni* che contiene il numero di giorni a partire dal 1-1-2000 (*numeroGiorni* = 0).
    - Il costruttore vuoto crea come data di default 1-1-2000.
    - Il costruttore Data(*a,m,g)* riceve una data e calcola anche i giorni. Verificare che i dati inseriti siano corretti utilizzando un vettore di appoggio con i giorni dei mesi e nel caso di febbraio, utilizzare anche il metodo privato *isBisestile(anno)* che restituisce true se l'anno è bisestile. Definire una funzione privata
    - Il metodo *setData(a,m,g) ei vari set()* impostano gli attributi e aggiornano i giorni verificandone la correttezza.
    - Utilizzando dei vettori che contengono i mesi dell'anno in lettere, creare il metodo *getDataEstesa()* che restituisce la data con il mese in parola.
    - Sapendo che il 1-1-2000 era un sabato, creare il metodo *GetGiornoSettima()* che restituisce il giorno della settimana per la data (lunedì, martedì,…)
    - Creare il metodo *compareTo*(d data) che restituisce 0 se le due date sono uguali, -1 se l'oggetto data su cui è invocato viene prima di d, 1 altrimenti
    - Creare il metodo *getDiff(d Data)* che restituisce il numero di giorni di differenza dell'oggetto su cui è invocato, rispetto alla data d
19) Usare la classe Data nella classe Nominativo

Vettori di Oggetti - Eccezioni Personalizzate – Enumerazioni

20) Creare la classe *Dado* e la classe *Secchiello* che contiene N dadi. Simulare il lancio dei dadi di un secchiello in grado di visualizzare i valori usciti e la loro somma. Deve essere possibile, per ogni dado stabilire il numero di facce di un dado.
21) Creare la classe *Punto* caratterizzata dai valori delle sue coordinate. Possiede, oltre ai set/get e al toString(), il costruttore senza parametri che inizializza le coordinate a 0 e un costruttore con due parametri. Creare quindi la classe *Figura* che ha come attributo il nome e il numero di vertici. Deve essere possibile aggiungere i suoi vertici e ottenere il valore del suo perimetro.
22) Si vuole gestire un concessionario monomarca di autovetture. A tal fine creare la classe *Auto* con gli attributi: modello, colore, prezzo; ha il costruttore con tutti e 3 gli attributi, il setPrezzo() e tutti i get oltre al toString(). I colori possibili sono solo: "rosso", "nero", "grigio", "bianco", per cui verificare che la stringa colore sia corretta, se no assegnare il colore "grigio". Si crei anche la classe *Concessionaria* con attributi un nome e un numero massimo di auto che può contenere, oltre al vettore di auto. Il costruttore ha come parametri il nome il numero massimo di auto. Alla concessionaria si possono aggiungere delle auto con il metodo add(auto) e devo poter stampare il parco macchine e il valore totale delle auto presenti in concessionaria. Inoltre dato un modello, vedere i colori delle auto presenti in concessionaria.
23) Scrivere una programma java per rappresentare dei giocatori di baseball. Per ciascun giocatore devono essere rappresentati il nome come stringa, un punteggio che rappresenta la bravura del giocatore in battuta e la sua età. Si devono quindi realizzare i seguenti metodi:
    - Giocatore(String nome, int eta) costruttore con due argomenti.
    - Giocatore(String nome, int eta, double valore) costruttore con tre argomenti.
    - String getNome() che ritorna il nome del giocatore.
    - double getPunteggio() che ritorna il punteggio del giocatore.
    - void setPunteggio(double nuovoPunteggio) che pone il punteggio del giocatore a nuovoPunteggio.
    - int getEta() che ritorna l'età del giocatore
    - int setEta() che modifica l'età del giocatore
    - String toString() che visualizza tutti i dati del giocatore

    Fai un menu in cui sia possibile inserire un nuovo giocatore, visualizzare i suoi dati.
24) Scrivere una classe java Squadra che ha come attributo un nome e un array di Giocatore dell'esercizio precedente ma con in più l'attributo ruolo che assume i valori di una enumerazione (BATTITORE, PRIMA BASE,…). Il costruttore crea una classe con un array di N_MAX_GIOCATORI vuoto. La classe possiede anche i seguenti metodi:
    - +addGiocatore(Giocatore g)
    - +void setPunteggio(String nome, double valore)
    - +double getPunteggioMedio() della squadra

- +passatoAnno() incrementa tutte le età
- +String toString()

25) Scrivere una classe java UtilityStringhe1 che contenga un metodo statico static void stampaIniziali(String nome, String cognome) che stampa a schermo le iniziali di nome e cognome. Scrivere una classe cliente ClienteUtilityStringhe1 che contiene un metodo main che prende due stringhe da tastiera (nome e cognome) e utilizzando il metodo stampaIniziali stampare le iniziali a schermo
26) Creare le classi necessarie per provare a vincere all'ambo su una ruota, raddoppiando sempre la giocata fatta. Quando si vince visualizzare la vincita (250 volte la puntata) e la somma complessiva giocata
27) Creare una classe *VettoriInteri* che:
    - Metodo costruttore con parametro un numero corrispondente alla dimensione desiderata che inizializza tutti gli elementi a 0
    - Metodo costruttore che riceve una stinga con gli elementi separati dalla "|" e carica il vettore con essi dimensionandolo della lunghezza giusta. Se nella stringa ci sono dei dati non corretti deve generare una eccezione *NumberFormatException*
    - Abbia un metodo che ritorna l'elemento minimo di un array, se il vettore è vuoto solleva l'eccezione personalizzata non controllata *ErroreVettoreVuoto*
    - Metodo che cerca un elemento nell'array e ritorna la posizione (se l'elemento non esiste solleva l'eccezione controllata personalizzata *ErroreElementoInesistente*)
    - Metodo che elimina la prima occorrenza nell'array, dato un numero (se l'elemento non esiste solleva l'eccezione controllata personalizzata *ErroreElementoInesistente*
    - Metodo toString() che restituisce gli elementi separati da "|"
28) Dichiarare una classe di nome Carburante che ha come attributi: *prezzo* float e un *tipo* che può essere "benzina", "diesel", "GPL", "metano" (utilizzare un vettore opportuno che contiene i valori possibili o un'enum). I metodi sono:
    - Costruttore vuoto che imposta tipo= "benzina" prezzo = 1.5
    - Costruttore con i due attributi (controlla il valore del prezzo)
    - Set/get di tutti gli attributi
    - equals(carburante c) che restituisce true se il carburante su cui è applicato il metodo ha lo stesso prezzo del carburante c passato come parametro
    - toString()

    Creare anche la classe Distributore che contiene più pompe di carburante. Ha come attributo il nome e l'indirizzo e il vettore con le pompe di carburante. Realizzare i metodi;
    - add(Carburante),
    - i set/get opportuni e il toString()
    - aumenta(tipo, percentuale) che aumenta il carburante di quel tipo della percentuale indicata
    - getStatistiche() che stampa il prezzo min, max e il medio tra i carburanti del distributore.
29) Scrivere una programma java che accetti in ingresso una serie di stringhe, le memorizzi in un oggetto e, al termine dell'inserimento, stampi la stringa più lunga immessa.
30) Un porto può accogliere N barche di diverso tipo a cui viene chiesto un TOT per ogni metro di lunghezza. Una barca si caratterizza per il C.F e il nome del proprietario, matricola, nome, dimensione in metri e anno di fabbricazione. Quando una barca entra in porto, gli viene assegnato un posto libero e dovrà pagare in base alla sua lunghezza (si farà metri x TOT) Realizzare oltre i metodi set e get opportuni, i seguenti metodi
    - Costruttore del porto di N posti inizialmente liberi con il prezzo al giorno passato per parametro
    - Metodo *nomeAffittuario*: dato in input un posto barca, restituire il nome dell'affittuario
    - Metodo *isLibero*: dato in input posto barca restituire se è libero o no
    - Metodo *affittaPosto*: dato in input posto barca se è libero assegnarlo al nuovo affittuario per la barca indicata, se no genera un'eccezione
    - Metodo *liberaPosto*: dato in input un posto barca se è occupato renderlo libero visualizzando l'importi dovuto, se no genera un'eccezione
31) Gestire dei parcheggi a pagamento che possono contenere solo un numero limitato di macchine e praticano un prezzo fisso all'ora (tipo 0,50€/h). Inoltre sono aperti solo in un determinato orario. Il sistema registra ad ogni ingresso di un'automobile targa e ora e all'uscita registra l'ora per quella targa e comunica l'importo di ingresso e l'ora di uscita. A fine giornata il sistema comunica al custode se ci sono ancora auto parcheggiate e calcola l'incasso totale. Gestire il tutto tramite menu: 0. esci 1.entra auto 2. esce auto 3.fine giornata
32) Si vuole realizzare un programma in Java per gestire la lista della spesa con il seguente menu:
    - `0-Esci`
    - `1-Nuova lista`
    - `2-Aggiungi prodotto alla lista`

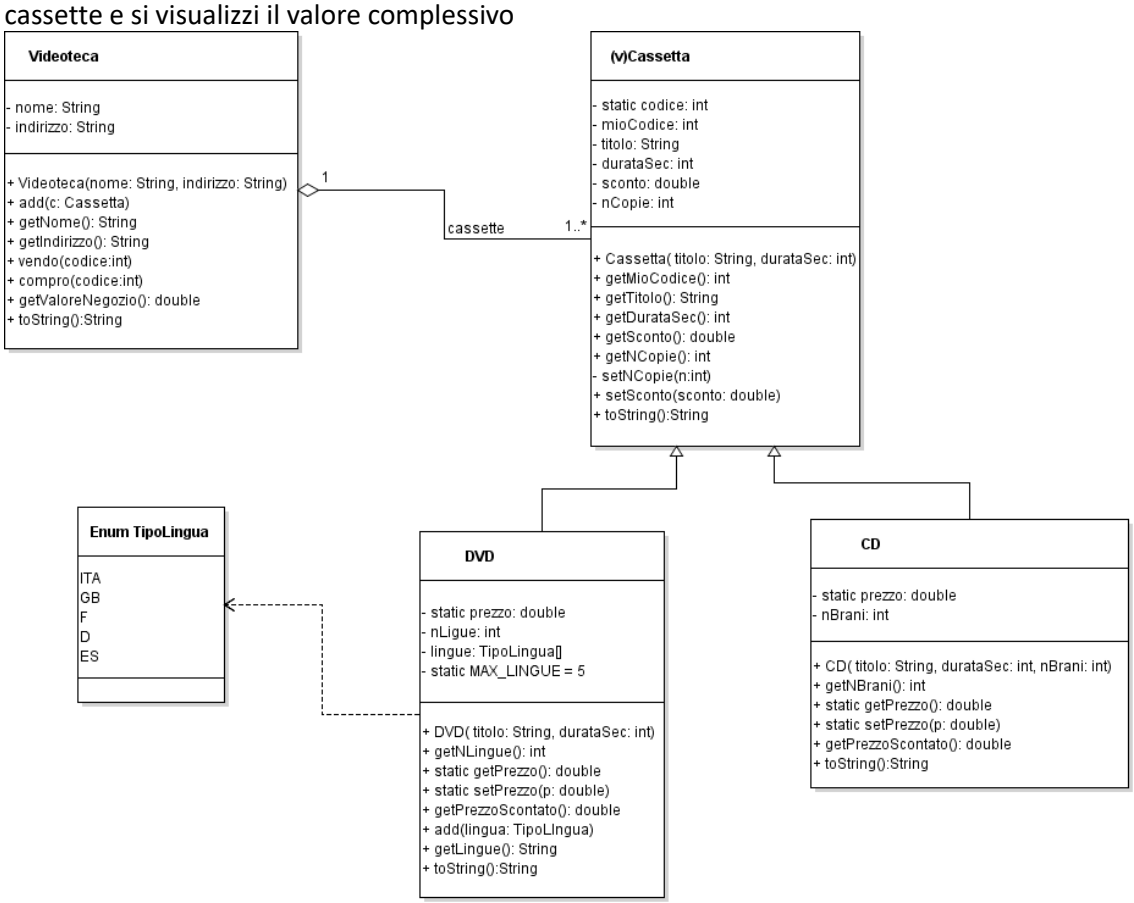
- `3-Compra prodotto al prezzo` *(Cerca il prodotto nella lista e se lo trova restituisce la quantità ancora da comprare (se ne ha comprati più di quelli richiesti restituisce comunque 0) RESITUISCE -1 se non trova il prodotto) si memorizza anche il prezzo di acquisto*
- `4-Visualizza lista della spesa (i prodotti ancora da compare)`
- `5-Vai alla cassa` *(visualizza il totale da pagare e gli eventuali prodotti non comprati)*

33) Scrivere una classe Proprietà che mantiene informazioni su proprietari di appartamenti. Ogni oggetto Proprietario deve contenere una stringa che indica il nome del proprietario, ed un vector di appartamenti di tipo stringa che possono contenere ciascuno l'indirizzo di un appartamento posseduto da quel proprietario. Le classi devono contenere:
    - costruttore che imposta il nome del proprietario ed il numero di appartamenti (ti serve per dimensionare il vector appartamenti, imposta l'incremento a 2)
    - metodo per assegnare un indirizzo di un appartamento ad un proprietario
    - metodo che restituisce l'indirizzo contenuto in una cella o "nessuno" se la cella è vuota
    - metodo che restituisce il numero di appartamenti inseriti
    - metodo che visualizza tutti gli appartamenti di un proprietario X inserito da tastiera
    - metodo che elimina un appartamento numero N di un proprietario X inserito da tastiera
    - metodo che visualizza tutti i proprietari con relativi appartamenti
    - metodo che elimina il proprietario X inserito da tastiera e i suoi appartamenti

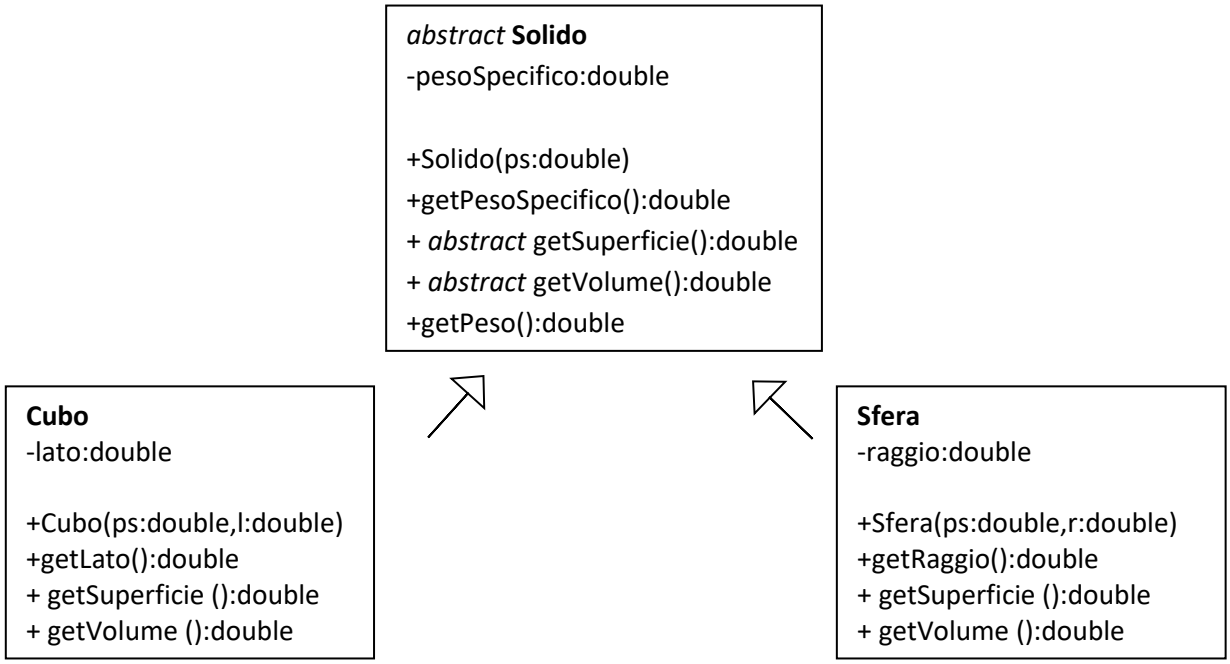
    Realizzare la classe `Inizio` con un menu per verificare il corretto funzionamento di tutte le funzioni.
34) Un importante locale commerciale della provincia di cuneo, chiede la realizzazione di un programma che gestisca le sue vendite. Il programma in questione deve permettere:
    A. Inserimento dei prodotti in vendita. Ogni prodotto deve avere: Codice(valore numerico), Descrizione e prezzo unitario.
    B. Registrazione vendite: di ogni vendita si conosce data, Codice venditore (Numero da 0 - 9), prodotto venduto, quantità venduta e come hanno pagato (CONTANTI, BONIFICO, CARTA DI CREDITO).
    C. Determinare quale venditore ha venduto più articoli.
    D. Determinare se un venditore (dato in input) ha realizzato una vendita, se sì mostrare tutti i dati degli articoli venduti, altrimenti messaggio di errore!
    E. Determinare il valore totale di tutte le vendite.
    F. Visualizzare la singola vendita con incasso più elevato pagata con carta di credito.
35) Creare un'applicazione JAVA per gestire un noleggio video. Il noleggio video possiede diversi tipi di prodotti. Tutti i prodotti possiedono: Titolo, Genere (Horror, Dramma, Avventura, Bambini), Prezzo noleggio al giorno, noleggiato (si/no). Si gestisca una lista di noleggi: NominativoCliente, Prodotto noleggiato, DataPrestito, DataReso. Creare un'applicazione (console) che:
    - Al momento della registrazione del reso visualizzi l'importo dovuto in base ai giorni di noleggio
    - Visualizza la lista dei film noleggiati
    - Dato un NominativoCliente visualizza i film noleggiati
    - Dato un NominativoCliente visualizza l'importo speso fino ad oggi
    - Visualizzare tutti i film dato il genere
    - Eliminare un cliente (dato il NominativoCliente)
    - Eliminare un film (dato il titolo)
36) Gestire i propri voti Scritto, Orale e Pratico per le varie materie e la data in cui sono stati assegnati. Gestire le eccezioni se i dati inseriti non sono corretti (per la data deve essere non successiva ad oggi e nel periodo corrente). Realizzare un programma che:
    - All'avvio viene richiesta la data di inizio e fine anno scolastico e di fine 1^ quadrimestre
    - Visualizzi i voti di una materia e ne calcoli la media
    - Calcolare la media globale di tutte le materia del primo e secondo periodo dell'anno
    - Media di S, O e P e media complessiva dei voti registrati fino ad ora di una materia
    - Cancellare i voti insufficienti
37) Una fabbrica di automobili produce 4 modelli di auto con prezzo di vendita: Modella A 9000 €, Modella B 10.500€, Modella C 14.500, Modella D 17.200€. Realizzare un programma che memorizzi in un file le varie vendite e visualizzi:
    - Il volume di vendita totale (in €)
    - La percentuale vendite di ogni modello
    - Dato in input un modello visualizzare quante auto sono state vendute

EREDITARIETÀ

38) Si scriva una classe Libro le cui istanze rappresentano libri. Ogni libro è caratterizzato dal numero di pagine. La classe deve contenere una variabile di istanza intera, numPagine, un costruttore che assegna al libro un numero specificato di pagine, ed un metodo pageMessage() che stampa il numero di pagine del libro.
    Scrivere quindi una classe Vocabolario che estende la classe Libro. La classe deve contenere una nuova variabile di istanza, numDefinizioni, un costruttore che assegna al vocabolario un numero di pagine e un numero di definizioni specificati, ed un metodo definitionMessage() che stampa un messaggio contenente il numero di pagine, il numero di definizioni ed il numero medio di definizioni per pagina del vocabolario.
    Scrivere infine un programma di prova per collaudare le classi e i metodi.
39) Si scriva una classe Lavoratore le cui istanze rappresentano lavoratori. Ogni lavoratore è caratterizzato da un nome, un livello (intero da 1 a 5) ed uno stipendio (mensile). La classe deve contenere un costruttore che assegna al lavoratore un nome ed un livello specificati, i metodi get e set per livello e stipendio, il get per il nome.
    Scrivere quindi una classe *LavoratoreConStraordinariPagati* che estende la classe Lavoratore. La classe deve contenere una nuova variabile di istanza, oreStraordinario, un costruttore che assegna al lavoratore con straordinari pagati un nome, un livello e un numero di ore di straordinario specificati, i metodi get/setOreStraordinari(). La classe deve inoltre contenere una variabile statica reale retribuzioneOraria, inizializzata a 10.0, corrispondente alla retribuzione di un'ora di straordinario (che si suppone uguale per tutti). Questo attributo di classe deve essere modificabile con il metodo set appropriato. La classe deve inoltre ridefinire il metodo getStipendio() per tenere conto della retribuzione delle ore di straordinario: alla retribuzione base (ereditata da Lavoratore) va sommata la retribuzione degli straordinari, ottenuta come retribuzione oraria dello straordinario per numero di ore di straordinario effettuate. Scrivere infine un programma di prova per collaudare le classi e i metodi.
40) Si vogliono gestire delle videoteche, che commerciano film in DVD e CD musicali. Il prezzo dei DVD è unico per tutti i DVD, lo stesso per i CD. Ogni cassetta ha un codice numerico gestito in automatico dal sistema, può avere o meno uno sconto da applicare al prezzo base (inizialmente 0), ha una durata complessiva in secondi e un numero di copie presenti in negozio (questa quantità inizialmente è 0, ma si modifica di 1 vendendo o comprando delle copie con i rispettivi metodi). I DVD sono caratterizzati dalle lingue possibili per il film (al massimo 5), che vengono aggiunte con il metodo *addLingua*() e sono visualizzabili con il metodo *getLingue().* I CD sono caratterizzati dal numero di brani presenti. Si vuole in ogni momento conoscere il valore complessivo delle cassette presenti in negozio con il metodo *getValoreNegozio()* (tenendo conto del numero di copie e degli eventuali sconti). Questa realtà è rappresentata dal seguente UML. Aggiungere le eccezioni necessarie.
    Realizzare una classe Test in cui si istanzia una videoteca con almento 2 DVD e 2 CD, si vendano e si comprino delle cassette e si visualizzi il valore complessivo



41) Realizza le seguenti classi e crea un main in cui si utilizza un vettore di Solidi e si istanziano 2 sfere e 2 cubi e si visualizzano tutte le caratteristiche



42) Le Figure si dividono in Cerchi, Quadrati e Triangoli. Ogni figura mette a disposizione i metodi draw() e erase() per rispettivamente stampare e cancellare la figura. Ipotizzando che draw() e erase() stampino semplicemente delle stringhe (esempio "Cerchio disegnato" e "Cerchio cancellato") che riportano l'operazione che si sta eseguendo realizzare le varie classi a partire dall'interfaccia Figura.

    Creare una nuova classe RandomShapeGenerator che mediante un metodo next() ritorni in modo random un'istanza di Circle, Square o Triangolo.

```
double x = Math.random();              // valore compreso [0..1)
double x = Math.random() * 10;         // valore compreso [0..10)
```

43) Definisci una interfaccia *Operazione* e le sue classi derivate (*Sub. Div, Mul*) in modo che da ciascuna sia possibile eseguire la corrispondente operazione aritmetica (di addizione, divisione, moltiplicazione) istanziando un oggetto di classe *Operazione*. Il metodo da implementare si chiama *getRisultato*() e ha i due operandi come parametri.
44) In un aerodromo si vuol mantenere informazioni sugli aeromobili. Di ogni aeromobile si vuole sapere la sigla (string) che lo identifica (univoca per ciascuno). Gli aeromobili si dividono in alianti e aerei a motore:
    - Gli alianti sono aeromobili caratterizzati da un numero (int) che ne descrive l'efficienza aereodinamica.
    - Degli aerei a motore si vuole sapere la potenza in CV del motore propulsore (double).

    Un oggetto può essere confrontabile con un altro secondo specifiche caratteristiche. Gli aeromobili dello stesso tipo devono essere confrontabili fra loro in modo da poter stabilire quale sia quello con prestazioni migliori: gli alianti si confrontano in base all'efficienza; gli aerei a motore in base alla potenza in CV.
    Crea un'interfaccia pubblica *Comparabile* contenente un metodo boolean *isSuperiore* (Comparabile x) per verificare se un oggetto (this) è superiore ad un altro x. Il metodo deve sempre restituire false se x è null o se gli oggetti coinvolti nel confronto non sono dello stesso tipo.
    Scrivi tre classi pubbliche: *Aeromobile, Aliante* e *Aereomotore* ciascuna con un metodo get pubblico per accedere all'attributo specifico della classe. Ogni classe deve fornire anche un costruttore, i metodi *eLoStesso* (se hanno la stessa sigla) e *toString*.

DIAGRAMMI UML

Progettare il diagramma delle classi per realizzare in Java il programma che risolve i seguenti problemi: realizza lo schema UML che rappresenta questa realtà indicando tutti gli attributi (anche quelli che derivano dalle relazioni, come per esempio i vettori), i metodi costruttori e i get/set che ritieni necessari MOTIVANDONE LA SCELTA, invece il toString() è sottinteso. Indica anche la cardinalità delle relazioni, dove è possibile.

45)

6 Un'agenzia di pratiche automobilistiche deve commissionare un software per il pagamento delle tasse annuali sui veicoli. Un professionista viene incaricato di progettare e implementare la gerarchia di classi che rappresentano i veicoli. Durante un'intervista al proprietario dell'agenzia emerge quanto segue:

- i veicoli possono essere motoveicoli o autoveicoli;
- per ogni veicolo è necessario memorizzare la targa, la marca, il modello, l'anno di immatricolazione e il numero di passeggeri consentito oltre al conducente;
- i motoveicoli sono sempre alimentati a benzina e sono caratterizzati dalla potenza espressa in HP: la tassa viene calcolata moltiplicando per la potenza un valore che a oggi vale 1,5 €/HP;
- gli autoveicoli tradizionali possono essere alimentati a benzina o a gasolio e sono caratterizzati dalla potenza espressa in HP: la tassa viene calcolata moltiplicando per la potenza un valore che a oggi vale 2,5 €/HP;
- per gli autoveicoli alimentati a gas, oltre alla potenza è necessario memorizzare il tipo gas (GPL o metano): questi autoveicoli non pagano nessuna tassa per i primi 5 anni dall'immatricolazione, trascorso questo periodo la tassa viene calcolata moltiplicando per la potenza un valore che a oggi vale 0,5 €/HP per il metano e 0,75 €/HP per il GPL;
- gli autoveicoli alimentati a gas idrogeno pagano una tassa che aumenta di 0,1 €/HP per ogni anno di vita del veicolo a partire da una tassa iniziale pari a 0 €/HP il primo anno di immatricolazione;
- come incentivo governativo gli autoveicoli elettrici non pagano alcuna tassa.

46)

7 Si intende gestire mediante un programma Java i vagoni che compongono un treno. Per ogni vagone si hanno alcuni attributi fondamentali:

- codice;
- peso a vuoto;
- azienda costruttrice;
- anno di costruzione.

Per i vagoni passeggeri si devono inoltre memorizzare:

- classe;
- numero di posti disponibili;
- numero di posti occupati;

mentre per i vagoni merci si devono memorizzare:

- volume di carico;
- peso massimo di carico;
- peso effettivo di carico.

Per la composizione di un treno è fondamentale la gestione del peso dei vagoni, che nel caso dei carri merci è di immediata determinazione, mentre per le carrozze passeggeri deve essere stimato considerando un peso medio per occupante di 65 kg (valore che potrebbe essere necessario modificare).
Dopo avere disegnato il diagramma UML delle classi della soluzione proposta e averlo implementato in linguaggio Java, codificare una classe Java *Treno* con uno o più metodi per l'aggiunta di vagoni: la classe dovrà prevedere un metodo che restituisca il peso complessivo del treno esclusa/e la/e motrice/i.

47) Un porto è caratterizzato da un nome, dalla località in cui si trova e dai posti barca che possiede. Il porto è suddiviso in moli numerati progressivamente con una tariffa giornaliera di ormeggio differente e una lunghezza e larghezza massima ammissibile per le barche, ciascun molo è formato da più posti barca, ciascuno dei quali ha un numero progressivo e può essere o meno occupato da una barca per tot giorni. Ogni barca possiede un nome, una nazionalità un proprietario e una lunghezza e larghezza. Deve essere possibile data una barca visualizzare i posti liberi in cui può essere ormeggiata con l'importo totale che dovrà pagare.

48) In un porto si affittano 100 posti barca di diverso tipo. Per ogni posto affittato si salvano la data inizio e fine dell'affitto (tipo GregorianCalendar) e la barca che lo occuperà. Una barca si caratterizza per il C.F e il nome del proprietario, matricola, nome, dimensione in metri e anno di fabbricazione. Un affitto si calcola moltiplicando il numero di giorni (affitto) per i metri della barca per un coefficiente € al metro. Realizzare oltre i metodi set e get opportuni, i seguenti metodi

- Costruttore del porto di 100 posti inizialmente liberi con il prezzo al metro passato per parametro

- Metodo *nomeAffittuario*: dato in input un posto barca, restituire il nome dell'affittuario
- Metodo *getPrezzo*: dato in input un posto barca restituire il prezzo
- Metodo *controlla*: dato in input posto barca restituire se è libero o no
- Metodo *affittaPosto*: dato in input posto barca se è libero affittarlo al nuovo affittuario per il periodo e per la barca indicata, se no genera un'eccezione
- Metodo *liberaPosto*: dato in input posto barca se è occupato renderlo libero visualizzando il totale dovuto, se no genera un'eccezione
- Metodi *toString()* per la barca, l'affitto e il porto

49) Un garage sotterraneo è formato da più piani ognuno con una lettera e un'altezza massima. In ogni piano ci sono più posti auto, ognuno con un numero identificativo e questi possono essere occupati o meno. Il costo orario è unico per tutto il garage. Quando un'auto arriva all'ingresso, le viene presentato l'elenco dei posti liberi per ciascun piano in cui può entrare (per l'altezza), quindi l'autista sceglie il posto e viene registrata l'ora di ingresso e fornito un biglietto con l'identificativo del posto. Per uscire, l'automobilista fornisce il biglietto e il sistema gli indica l'importo da pagare e libera il posto che occupava.
50) Un appartamento è composto da una o più stanze, ciascuna delle quali ha una lunghezza e una larghezza. Un appartamento è posseduto da uno o più persone. Un palazzo ha un certo numero di piani ed è composto da uno o più appartamenti. Gli appartamenti di un palazzo hanno un valore al metroquadro diverso per ogni piano. Deve essere possibile conoscere il valore di un appartamento e il valore immobiliare complessivo posseduto da una persona.
51) Una pagina in HTML ha un corpo (body) formato da più elementi. Ogni elemento è caratterizzato da un TAG e, nel caso di paragrafi, titoli, …, da un contenuto testuale. Inoltre ogni tag può avere più attributi caratterizzati da un nome e un valore.
52) In un gioco i personaggi vengono costruiti assemblando varie parti del corpo: arti, al massimo 4, di cui si sceglie il colore e una percentuale di forza per sferrare i colpi, gambe massimo 2, di cui si sceglie il colore e la lunghezza. Inoltre ci sono delle armi che possono essere usate dai vari personaggi, e sono caratterizzate da un nome, una potenza e un percentuale di ricarica. Ogni mostro può avere al massimo un'arma per arto. Inoltre i mostri possono avere dei vestiti: un cappello, una maglia, un pantalone. Di ciascun abbigliamento si può scegliere il colore. Creare una classe test che visualizzi il mostro creato e la sua forza totale negli arti e nelle armi.
53) La WIND ha un proprio piano tariffario per la telefonia business che prevede per i suoi clienti uno dei possibili contratti: START, PLUS, EVO con costo mensile rispettivamente di 18, 35 e 65 euro. I contratti possono prevedere opzioni come: 4G, Internet gratis nei week-end, … ognuno con il suo costo. Solo i maggiorenni possono sottoscrivere il contratto ed è possibile inserire nel contratto fino a 3 numeri telefonici di colleghi (anche di altre compagnie) con cui parlare gratis. I contratti si rinnovano, salvo disdetta, ogni anno e deve essere possibile inviare la fattura ed un eventuale sollecito di pagamento se questo non avviene entro 30 giorni dall'invio della fattura.
54) L'azienda SchoolRobot produce robot per l'industria metallurgica. Tutti i robot dispongono di un microprocessore (di cui si vuole conoscere velocità di clock e modello) e di memoria RAM (di cui si vuole conoscere la dimensione in MB). Una tipologia di robot serve per il taglio laser di lamiere (tipo A), una per la saldatura automatica (tipo B) ed una per la tornitura a controllo numerico (tipo C). L'azienda ha necessità di memorizzare gli ordini dei clienti. Per ogni ordine occorre memorizzare la denominazione del cliente, indirizzo, numero d'ordine, data ordine e l'elenco dei robot ordinati. Per i robot di tipo A occorre indicare tipo del processore, quantità di memoria, spessore di taglio in mm, precisione (decimi di millimetro); per i robot di tipo B occorre indicare tipo del processore, quantità di memoria, nr punti di saldatura al secondo; per il il tipo C occorre indicare tipo del processore, quantità di memoria, materiali lavorabili (acciaio, alluminio, ferro). Si proceda a:
    - implementare tutte le classi necessarie in codice java compresa la classe per gestire le eccezioni;
    - implementare il metodo public void addCliente() in grado di richiedere all'utente l'inserimento da tastiera dei dati di un cliente ed aggiungerlo all'elenco clienti
55) Si vogliono gestire le contravvenzioni effettuate dai vari comandi dei vigili di un comune. Di ogni contravvenzione interessa il veicolo a cui è stata effettuata, il vigile, il numero di verbale e il luogo in cui è stata emessa. Di ogni vigile interessa il nome, il cognome ed il numero di matricola. Di ogni comando interessa il nome della caserma, l'indirizzo, il numero di telefono e i vigili che fanno parte di esso. Di ogni veicolo interessa la targa. Esistono solamente due categorie di veicoli: automobili e motocicli. Delle automobili interessa la potenza in kilowatt, dei motocicli il numero di telaio. Scrivere in Java il codice della classe che permette l'inserimento di una nuova contravvenzione e quella relativa ad un Autoveicolo.
56) Si vogliono gestire le riparazioni che effettuano le varie officine meccaniche. Di ogni officina interessano nome, indirizzo, numero dipendenti, e quali dipendenti lavorano in essa. Dei dipendenti interessano: nominativo, anni di

servizio e numero telefono. Quando un veicolo arriva in un officina per una riparazione, bisogna memorizzare ora e data di accettazione e quando viene riconsegnato, ora e data di riconsegna, costo della riparazione e descrizione dei lavori effettuati. Dei veicoli interessa memorizzare: modello, tipo (autoveicolo, motoveicolo, autocarro, …), targa, anno di immatricolazione e i dati del proprietario (CF, indirizzo, nominativo, telefono). Ogni officina è diretta da un direttore di cui interessa conoscere anche la data di nascita, ma non gli anni di servizio.

57) Si deve realizzare un sistema informatico per gestire le informazioni necessarie per amministrare dei condomini. Un condominio è ubicato in una via di una città ed è formato da appartamenti, caratterizzati ciascuno dalla metratura in mq e dai millesimi di proprietà, in base ai quali vengono ripartite le spese. Ogni appartamento appartiene ad un proprietario ed è occupato da un inquilino (le due figure possono anche coincidere) di cui si vuole memorizzare il nominativo e il telefono e, inoltre per gli inquilini il numero di persone nel nucleo familiare, mentre per il proprietario l'indirizzo a cui mandare le comunicazioni. Nella gestione del condominio le spese vengono ripartite tra proprietari e inquilini dei vari appartamenti, che devono fare dei versamenti in una certa data e di un certo importo. Di ogni versamento deve essere possibile sapere per quale appartamento è stato fatto e se è relativa a una spese per il proprietario o per l'inquilino.

58) Una Società Aeroportuale commissiona lo sviluppo di un software per la gestione dei controlli passeggeri e merci nei punti di dogana presenti nei vari Aeroporti Internazionali. Un passeggero arriva ad un punto di controllo, un addetto gli controlla il passaporto e l'eventuali merci trasportate (per es. un cellulare nuovo e dei liquori) e registra i dati. Se a seguito del controllo qualcosa non va si può trattenere il passeggero o la merce o far pagare un dazio. Dell'addetto si vuole conoscere il nominativo e il numero di tesserino di identificazione. Degli aeroporti il nome e la città; dei punti di controllo il gate (è un numero) in cui si trova.
    Si richiede la memorizzazione, per ogni controllo effettuato, dei dati relativi a:
    - passeggero: nominativo, nazionalità, N. passaporto o carta d'identità;
    - merci che il passeggero trasporta, caratterizzate ciascuna da: categoria di appartenenza (generi alimentari, strumentazione elettronica, farmaci), dazio doganale, descrizione, quantità dichiarata;
    - punto di controllo, addetto al controllo, data e ora inizio controllo, esito (nessuna segnalazione, merce respinta, fermo del passeggero), dazio doganale eventualmente da pagare(il totale degli eventuali dazi dovuti)

Progetto autonoleggio http://www.labsquare.it/?p=5462

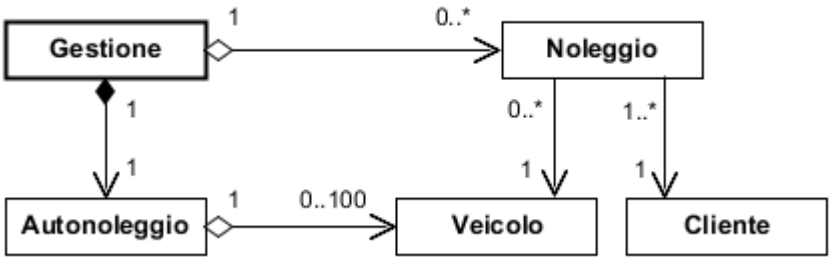
Si vuole realizzare un programma in Java per la gestione del noleggio dei veicoli di un **autonoleggio**. Tra i dati che devono essere registrati di ogni **veicolo**, ci sono: un codice identificativo, la marca della casa costruttrice, il modello, il numero di posti, la targa. In particolare si richiede che il codice identificativo sia di tipo numerico e sia gestito dal sistema in auto-incremento ogni volta che un nuovo veicolo viene aggiunto all'autonoleggio. I veicoli da gestire sono una cinquantina. Quando un veicolo viene noleggiato bisogna registrare i dati del noleggio comprensivi dei dati del **cliente** che ha effettuato il **noleggio** e del veicolo scelto. I noleggi possono essere prenotati in anticipo, con l'indicazione del periodo (date di inizio e di scadenza del noleggio). Per semplificare il problema, supporremo che l'autonoleggio sia interessato a mantenere a sistema le registrazioni dei dati del noleggio e del cliente corrispondente, solo per la durata del noleggio stesso e che, quindi, al rientro nell'autonoleggio del veicolo noleggiato, questi dati debbano essere cancellati dall'applicazione che non potrà così più accedervi. Tuttavia, l'archiviazione persistente dei dati che riguardano ciascun noleggio deve essere realizzata memorizzando i dati in un file di testo nel formato CSV, utilizzando il punto e virgola come carattere delimitatore (ndr. un'introduzione ai file CSV è presente in quest'articolo: link). Ciò renderà lo storico dei noleggi disponibile ad applicazioni esterne, quali un foglio di calcolo. L'applicazione, infine, deve permettere di esportare tutti i veicoli dell'autonoleggio e di importarne di nuovi, attraverso un file di testo nel formato CSV. In definitiva, le funzionalità che devono essere disponibili nell'applicazione sono:

1. Gestione noleggi:
    a. Aggiunta di un nuovo noleggio/prenotazione di un veicolo da parte di un cliente, con cui si registrano le date di inizio e fine noleggio, i dati del cliente e un riferimento al veicolo noleggiato (*vincolo progettuale*

per supposti sviluppi futuri: come riferimento al veicolo utilizzare il *codVeicolo* piuttosto che il riferimento Java dell'oggetto Veicolo).
    b. Rientro del veicolo nell'autonoleggio, con cui il noleggio corrispondente non risulterà più disponibile per il programma (viene eliminato) e tutti i dati del noleggio (data di inizio, data di scadenza, data in cui il veicolo rientra e i dati del cliente e del veicolo corrispondenti) verranno archiviati in maniera persistente in un file di testo nel formato CSV utilizzando il punto e virgola come carattere delimitatore.
2. Visualizzazioni:
    a. Visualizzazione dell'elenco di tutti i veicoli che appartengono al parco veicoli dell'autonoleggio.
    b. Visualizzazione dell'elenco di tutti i veicoli disponibili per il noleggio in un periodo di tempo indicato.
    c. Visualizzazione dell'elenco di tutti i noleggi/prenotazioni in corso.
3. Gestione veicoli:
    a. Aggiunta di un nuovo veicolo al parco veicoli dell'autonoleggio.
    b. Cancellazione di un veicolo dal parco veicoli dell'autonoleggio.
4. Gestione persistenza su file:
    a. Salvataggio su file di tutti i dati caricati a sistema, in modo che essi possano essere ripristinati all'avvio dell'applicazione.
    b. Importazione dei veicoli da un file CSV, con cui è possibile aggiungere nuovi veicoli al parco dei veicoli dell'autonoleggio.
    c. Esportazione di tutti i veicoli dell'autonoleggio in un file CSV.
5. Uscita dal programma, con cui tutti i dati del sistema vengono salvati su un file (lo stesso del pungo 4.a).

Soluzione proposta

Diagramma UML delle classi



La figura precedente mostra il diagramma UML delle classi per la soluzione che viene qui proposta. Le entità elementari individuate analizzando il problema sono: *Veicolo*, *Cliente* e *Noleggio*. I veicoli che fanno parte dell'autonoleggio (da 0 fino a 100) possono pensarsi **aggregati** nella classe *Autonoleggio*. Quest'ultima va a **comporre** la classe *Gestione*, la quale si occupa di gestire per l'autonoleggio il modo in cui via via è possibile **associare** i noleggi con i veicoli e i clienti. Ad esempio, quando si inserirà un nuovo noleggio bisognerà associare ad esso un cliente e un veicolo, verificando che il veicolo richiesto sia disponibile per il noleggio nel periodo indicato. Il **main()** è contenuto nella classe *Gestione*, che per questo è stata evidenziata in grassetto, ed è in essa che i noleggi vengono **aggregati** man mano che vengono istanziati.

Il diagramma riporta anche la **navigabilità** di ciascuna associazione rappresentata con la **freccia** che, si ricorda, è il modo in cui esprimere in UML la ***responsabilità*** della *classe di partenza* di fornire un collegamento, ossia un riferimento o comunque un modo, che permetta di raggiungere da un oggetto della *classe di partenza* gli oggetti ad esso associati della *classe di arrivo.*

Per la definizione delle associazioni diverse dalle **composizioni** ed **aggregazioni** descritte sopra, quelle che legano le classi Noleggio, Veicolo e Cliente, si può trovare che ciascun noleggio si associa ad un cliente e la navigabilità è in un solo verso, quello indicato dalla freccia, in quanto il problema non ci richiede di poter risalire a quali sono i noleggi di un fissato cliente a partire direttamente da Cliente. Ciascun noleggio, infine, deve essere associato ad un solo veicolo e, vale la pena sottolineare, che ciascun veicolo può associarsi a più noleggi, in quanto, potendo i noleggi essere prenotati in anticipo, in ogni momento ciascun veicolo può avere più noleggi ad esso associati (prenotati). Per la navigabilità in un solo verso di questa associazione valgono le stesse considerazioni fatte per Cliente e Noleggio.

Inner Class (composizione)

59) Nell'ambito di un programma di geometria, si implementi la classe Triangolo, il cui costruttore accetta le misure dei tre lati. Se tali misure non danno luogo ad un triangolo, il costruttore deve lanciare un'eccezione. Il metodo getArea restituisce l'area di questo triangolo. Si implementino anche la classe *Triangolo.Rettangolo*, il cui costruttore accetta le misure dei due cateti, e la classe *Triangolo.Isoscele*, il cui costruttore accetta le misure della base e di uno degli altri lati.

Si ricordi che:

- Tre numeri a, b e c possono essere i lati di un triangolo a patto che a<b+c, b<a+c e c<a+b.
- L'area di un triangolo di lati a, b e c calcolatela con la formula di Erone,

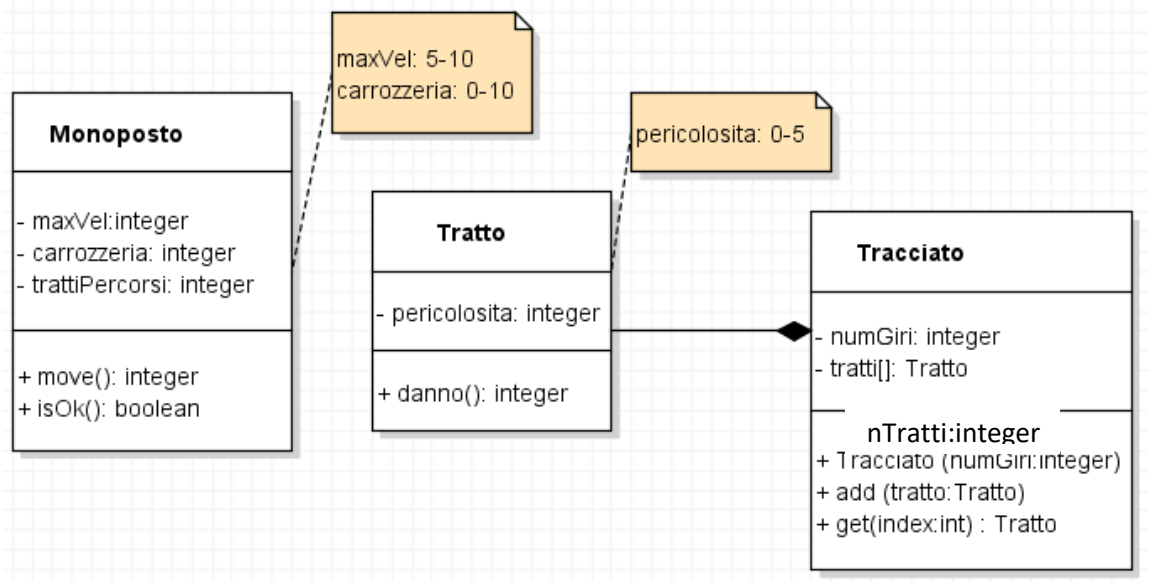
```
Triangolo x = new Triangolo(10,20,25);
Triangolo y = new Triangolo.Rettangolo(5,8);
Triangolo z = new Triangolo.Isoscele(6,5);

System.out.println(x.getArea());
System.out.println(y.getArea());
System.out.println(z.getArea());
```

Output dell'esempio d'uso:

```
94.9918
19.9999
12.0
```

60) Creare le classi rappresentate dal seguente diagramma UML per realizzare una gara di automobili:



- Il metodo costruttore delle monoposto inizializza in modo casuale i parametri carrozzeria e maxVelocita
- Il metodo move() fa avanzare di un numero n casuale rispetto alla velocità max possibile la monoposto, facendole percorrere n tratti, in ciascuno dei quali può ricevere un danno
- Il metodo isOk() dice se l'automobile è ancora "viva" (attributo carrozzeria)
- Il costruttore di tratto imposta in modo casuale la pericolosità e il metodo danno() restituisce un possibile danno nel range stabilito dalla pericolosità
- il costruttore di Tracciato crea un percorso formato da nTratti aggiunti con valori casuali di pericolosità
- Realizza la classe Main che simula la gara tra due automobili

61) Scrivere una programma java per rappresentare delle cisterne. Per ciascuna cisterna si deve rappresentare un identificativo numerico univoco per la cisterna, una capacità massima e lo stato attuale della cisterna (quantità di materiale contenuto nella cisterna). Si devono quindi realizzare i seguenti metodi:

- `Cisterna(int identificativo, int capacita)` costruttore con due argomenti pone lo stato della cisterna a 0.
- `Cisterna(int identificativo, int capacita, int stato)` costruttore con tre argomenti.
- `int getId()` che ritorna l'identificativo della cisterna.
- `int getCapacita()` che ritorna la capacità massima della cisterna.
- `int getStato()` che ritorna lo stato attuale della cisterna.
- `boolean aggiungiValore(int quantita)` che aggiunge il valore quantità allo stato della cisterna. Se lo stato eccede la capacità lo stato deve essere posto pari alla capacità della cisterna e viene sollevata

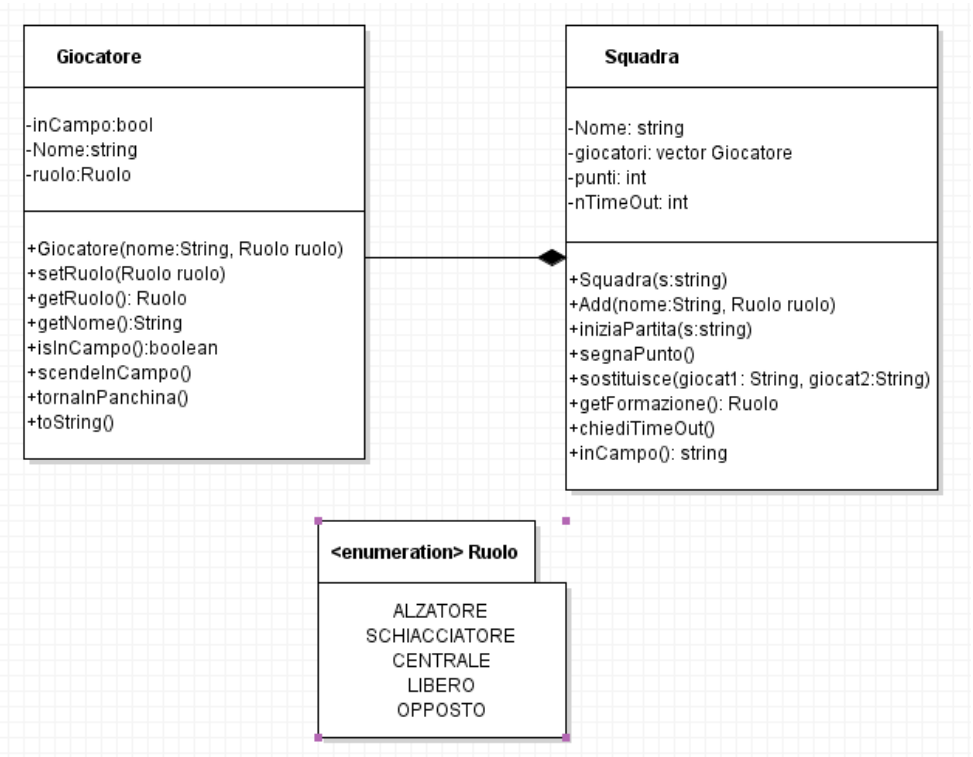
un'eccezione indicante cisterna piena ed il metodo ritorna il valore false. Altrimenti il metodo ritorna il valore true.

- `int togliValore(int quantita)` che preleva il valore quantità dallo stato della cisterna e restituisce lo stato attuale. Se lo stato risulta minore di quantità il metodo restituisce come valore lo stato attuale = 0 e e viene sollevata un'eccezione indicante cisterna vuota.
- `boolean equivalente(Cisterna c)` che ritorna true se l'oggetto di invocazione è uguale alla cisterna c. Due cisterne sono equivalenti se hanno la stessa capacità residua: capacità massima - stato attuale.

Si realizzi inoltre una classe di massimo 10 cisterne che ne permetta l'inserimento di cisterne con identificatore diverso, la modifica della cisterna richiesta e la classe `Inizio` con un menu per verificare il corretto funzionamento di tutte le funzioni

62) Una rubrica contiene informazioni (nome, indirizzo,numero telefonico) su un certo numero di persone (per esempio 5) prestabilito (le informazioni sono reintrodotte nel metodo main()). L'utente dovrà fornire all'applicazione un nome da riga di comando e l'applicazione dovrà restituire le informazioni relative alla persona. Se il nome non è fornito, o se il nome immesso non corrisponde al nome di una persona preintrodotta dall'applicazione, deve essere restituito un messaggio significativo.

63) Realizzare un progetto Java che implementi la classe Giocatore e la classe Squadra (di pallavolo) descritti dall'UML sottostante



- il costruttore della squadra riceve il Nome della squadra e i suoi componenti con i relativi ruoli sotto forma di stringa divisa da "," (Esempio: Nome: Grizzlies, Componenti squadra: Dentello,Donati,Bagnis,Ribero,Nazari, Peirone)
- iniziaPartita() riceve la stringa con il nome dei giocatori che mette in campo e azzera i punti
- segnaPunto() la squadra vince un punto durante una partita;
- sostituisce() la squadra sostituisce un componente con un un'altra persona;
- chiediTimeOut()la squadra richiede un time-out (al massimo due);

Il programma principale deve prevedere di prendere in input due squadre che si affrontano nella partita, complete di nome e componenti. Successivamente deve iniziare un ciclo nel quale chiede all'utente l'evento che capita – i possibili eventi sono:

- P – una squadra segna un punto – il programma deve chiedere quale squadra ha vinto il punto prendendo i input un numero tra 1 e 2;
- S – una squadra sostituisce un componente – il programma deve chiedere quale squadra ha chiesto la sostituzione, la persona da sostituire e quella che la sostituisce – le stringhe devono essere nella forma "Dentello-Cerato";
- T – una squadra chiede un time-out – il programma deve chiedere quale squadra chiede il time-out – deve visualizzare se viene accettato o rifiutato;

Il programma finisce quando una delle due squadre raggiunge i 25 punti e ha più di due punti di vantaggio sull'avversario.

64) Un giocatore è caratterizzato da un nome ed un età. Un giocatore di pallavolo è un giocatore che in più è caratterizzato da un ruolo e da un valore espresso da un punteggio. Si devono quindi realizzare i seguenti metodi:

- `Giocatore(String nome, int eta)` costruttore con due argomenti.
- `GiocatorePallavolo(String nome, int eta, double punteggio, Ruolo ruolo)` costruttore

- `String getNome()` che ritorna il nome del giocatore.
- `double getPunteggio()` che ritorna il punteggio del giocatore.
- `void setPunteggio(double nuovoPunteggio)` che pone il punteggio del giocatore a nuovoPunteggio.
- `int getEta()` che ritorna l'età del giocatore.
- `String toString()` che restituisce tutti i valori degli attributi separati da ';'
- `boolean isMigliore(GiocatorePallavolo g)` che ritorna true se l'oggetto di invocazione è migliore del giocatore g passato come parametro, false altrimenti. Un giocatore è migliore di un altro se ha un punteggio più alto. A parità di punteggio il giocatore con l'età più bassa è considerato migliore.

Si realizzi inoltre una classe `Squadra` in cui si caricano i giocatori del torneo in un vettore di massimo 15 (in ogni momento sono caricati `nGiocatori`). Questa classe ha anche i metodi:

- `addGiocatore(String nome, int eta, double punteggio, Ruolo ruolo)` che aggiunge un nuovo giocatore.
- `double getPunteggioMedio()` che restituisce il punteggio medio dei giocatori della squadra
- `passatoAnno()` che incrementa di 1 tutte le eta dei giocatori

Nel main creare un programma che testi le varie funzionalità tra cui l'elezione del migliore giocatore del torneo.