PROGRAMMAZIONE (9 CFU) – A.A. 2016-2017	ES. 05

Inserire nel package **it.unipa.community.nomecognome.prg.n05.esX** le seguenti applicazioni.

1) In un'applicazione per la gestione di una videoteca, i clienti sono memorizzati in oggetti della classe Persona. Ogni Persona è caratterizzata da una datadiNascita, nome, cognome, codiceFiscale, indirizzo, citta, cap. Creare una classe Abbonato che estenda la classe Persona memorizzando in un'opportuna variabile d'istanza sconto la percentuale di sconto a cui l'Abbonato ha diritto su ogni acquisto effettuato. Prevedere opportuni metodi per l'incapsulamento di questa variabile.
   Creare inoltre una classe AbbonatoPremium che, oltre ad aver diritto ad uno sconto, ha diritto ad un acquisto gratuito ogni volta che accumuli una spesa complessiva superiore a 100€. Scrivere una classe per testare le classi Abbonato e AbbonatoPremium.

2) Creare una gerarchia di classi che possa rappresentare le seguenti entità: Persona, Professore, Studente, StudenteTriennale e StudenteMagistrale.
   Ogni Professore ha una dataAssunzione, un ruolo (Ricercatore, Professore Associato o Professore Ordinario), un dipartimento di appartenenza (es. DIID, DICAM, ...). Ogni Professore percepisce un salario (prevedere quindi i metodi getSalario() e setSalario()).
   Ogni Studente ha una dataIscrizione, una matricola, un corsoDiLaurea a cui è iscritto. Ogni Studente paga un contributo d'Iscrizione al corso.
   Uno StudenteTriennale deve conseguire 180 CFU e proviene da una scuolaSuperiore (una stringa per memorizzare la scuola di provenienza). Uno StudenteMagistrale deve conseguire 120 CFU e proviene da un corsoTriennale (una stringa per memorizzare il corso di laurea triennale di provenienza).
   Prevedere opportuni metodi per l'incapsulamento dei dati.
   Laddove possibile, riutilizzare classi sviluppate in esercitazioni precedenti (per es. Date e Persona). In ogni classe prevedere il metodo toString() cge restituisce una stringa descrittiva dell'oggetto (es. per Persona: "*<nome> <cognome> <dataDiNascita> <indirizzo>*", per Studente: "*<matricola<nome> <cognome> <dataDiNascita> <indirizzo> <corsoDiLaurea>...*", etc). (NB. Con la notazione <nome> si indica il valore che la variabile nome assume).
   Scrivere un'applicazione che consenta di testare le classi precedentemente descritte.

3) Creare una classe Point2D per rappresentare punti geometrici in 2D. Point2D incapsula le coordinate x e y. Oltre ai soliti metodi per l'incapsulamento, prevedere un metodo toString() che descriva il punto come segue:"(<x>, <y>)".
   Creare una classe Line che permetta di rappresentare una retta passante per due punti (Line memorizza due punti in oggetti della classe Point2D). Prevedere un metodo che, dato in input un Punto2D, verifichi se il punto appartiene alla retta o meno.

   Retta per due punti: $\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$

4) Creare una classe Point3D che estende la classe Point2D per incapsulare le coordinate x, y, z. Prevedere un metodo toString() che descriva un punto 3D come: "(<x>, <y>, <z>)".
   Creare una classe Plane che permetta di rappresentare un piano passante per tre punti (utilizzare Point3D). Prevedere un metodo che, dato in input un Punto3D, verifichi se il punto appartiene al piano o meno.

   Piano per tre punti:

$$ax + by + cz + d = 0$$

$$a = \begin{vmatrix} y_2 - y_1 & z_2 - z_1 \\ y_3 - y_1 & z_3 - z_1 \end{vmatrix}$$

$$b = -\begin{vmatrix} x_2 - x_1 & z_2 - z_1 \\ x_3 - x_1 & z_3 - z_1 \end{vmatrix}$$

$$c = \begin{vmatrix} x_2 - x_1 & y_2 - y_1 \\ x_3 - x_1 & y_3 - y_1 \end{vmatrix}$$

$$d = -(ax_1 + by_1 + cz_1)$$

5) Costruire una gerarchia di classi che consenta di rappresentare le seguenti entità: Shape, Circle, Rectangle, Square.
Shape è caratterizzato da una stringa rappresentante un colore (color) e da una variabile boolean che indica se è una figura piena o no (filled).
Circle è caratterizzato da un raggio (radius). Rectangle è caratterizzato da width e length. Square è un caso particolare di Rectangle in cui base e altezza sono uguali.
In ogni classe, prevedere opportuni metodi per l'incapsulamento dei dati e metodi per determinare area e perimetro (nel caso di un Circle il perimetro sarà la circonferenza). Prevedere anche i metodi toString().

6) Si costruisca una gerarchia di classi per rappresentare veicoli su terra considerando le seguenti entità: Veicolo, VeicoloAMotore, Ciclomotore, Automobile, Bicicletta.
Un Veicolo è caratterizzato da una posizione, una velocità iniziale e un'accelerazione (per rappresentarli si usi una classe Vettore2D che rappresenta un vettore tramite un Punto2D e che fornisce i metodi modulo e fase).
In base ai valori di velocità e accelerazione, un Veicolo segue la legge di moto uniformemente accelerato:

$$\vec{x} = \vec{x}_0 + \vec{v}\Delta t + \frac{1}{2}\vec{a}\Delta t^2$$

(si utilizzi il metodo muovi(double deltaT) dove deltaT rappresenta la variazione di tempo durante cui il veicolo si muove).
VeicoloAMotore è un Veicolo caratterizzato da un motore con una cilindrata predefinita.
Ciclomotore è un VeicoloAMotore caratterizzato da due Ruote (si utilizzi la classe Ruota di un'esercitazione precedente).
Automobile è un VeicoloAMotore caratterizzato da quattro Ruote.
Bicicletta è un Veicolo caratterizzato da due Ruote.
Scrivere un'applicazione che consenta di testare la gerarchia delle classi e di simulare il movimento nel tempo di una Bicicletta, un'Automobile e un Ciclomotore avviati tutti allo stesso istante di tempo.

NOTE PER COMPILAZIONE E TEST A RIGA DI COMANDO IN AMBIENTE LINUX:

Creare una cartella col proprio cognome sulla scrivania e i relativi file sorgenti al suo interno.
*Aprire una finestra di **terminale** e digitare:*
`cd Desktop/cognome` *oppure* **`cd Scrivania/cognome`** *(si posiziona nella directory)*
*Creare i file sorgente con **gedit** e salvarli nella propria directory.*
Digitare:
`javac nomeClasse.java` *(compila e genera il bytecode)*
`java nomeClasse` *(esegue il bytecode sulla JVM)*