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** che memorizzi una lista di oggetti e abbia funzionalità per aggiungere nuovi abbonati, gestire il costo degli acquisti in base al tipo di abbonato e stampare lo stato degli abbonati.

2. Si costruisca una gerarchia di classi per rappresentare veicoli su terra considerando le seguenti entità: **Veicolo, VeicoloAMotore, Motociclo, 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 le componenti x e y (double) e fornisce opportuni costruttori e metodi set e get.
   In base ai valori di velocità e accelerazione, un **Veicolo** segue la legge di moto uniformemente accelerato:
   $$x = x_0 + v_{0x} * t + a_x * t * t$$
   $$y = y_0 + v_{0y} * t + a_y * t * t$$
   (si scriva un metodo *muovi(double t)* dove *t* rappresenta la variazione di tempo durante cui il veicolo si muove).
   VeicoloAMotore è un **Veicolo** caratterizzato da un motore con una cilindrata predefinita e da una targa (St*ring*). **Bicicletta** è un **Veicolo** caratterizzato dal modello (*String*). **Automobile** e **Motociclo** sono **VeicoliAMotore** che non hanno caratteristiche aggiuntive ma che predisponiamo per usi futuri.
   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.

3. Si consideri la gerarchia di classi **Shape, Circle, Rectangle, Square** (esercitazione 7). Si modifichino le classi in modo da lanciare opportune eccezioni quando gli oggetti delle varie classi vengono inizializzati con valori negativi (**IllegalArgumentException**). Si scriva un'applicazione che istanzi oggetti delle suddette classi e gestisca le eventuali eccezioni.

4. Si consideri la classe **IntegerSet** (esercitazione 5) e la si modifichi prevedendo il lancio di un'eccezione **IndexOutOfRangeException** se si tenta di inserire un intero che non appartiene al range [0, 99]. Si scriva un'applicazione che istanzi oggetti della suddetta classe e gestisca le eventuali eccezioni.

5. Si consideri la classe **Stack** (esercitazione 5) e la si modifichi in modo da lanciare un'eccezione personalizzata di tipo **MyStackOverflow** in caso di tentativo di inserimento a stack pieno.

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

Digitare per ciascuna classe:
`javac nomeClasse.java` *(compila e genera il bytecode)*
Digitare per la classe che contiene il main:
`java nomeClassePrincipale` *(esegue il bytecode sulla JVM)*