1) Modificate l'esercizio 4 dell'esercitazione 4 utilizzando un singleton che gestisce tutto l'input da tastiera. Questa classe dovrà disporre del metodo **getInstance** che restituisce l'unica istanza dell'oggetto (che al suo interno avrà uno Scanner) e di metodi per leggere da tastiera valori di vario tipo eventualmente anche con la stampa di un messaggio di prompt da passare come parametro (ad esempio "Inserisci nome: ")

2) Create la classe **IntegerSet** (insieme di interi). Ogni oggetto di **IntegerSet** può memorizzare interi fra 0 e 99. Un insieme è rappresentato internamente da un array di booleani. L'elemento dell'array **a[i]** è **true** se l'intero **i** è contenuto nell'insieme. L'elemento dell'array **a[j]** è **false** se l'intero **j** non è contenuto nell'insieme. Il costruttore di default inizializza un insieme all'insieme vuoto, il cui array contiene soltanto **false**.
   Scrivete i metodi che effettuano le comuni operazioni sugli insiemi. Per esempio scrivete la funzione **unionOfIntegerSet** che crea un terzo insieme che è l'unione di due insiemi esistenti: un elemento del terzo insieme è **true** se è presente in almeno uno dei due insiemi origine. Scrivete il metodo **intersectionOfIntegerSet** che crea un terzo insieme che è l'intersezione di due insiemi esistenti: un elemento del terzo insieme è **true** se è presente in entrambi gli insiemi origine. Scrivete la funzione membro **insertElement** che inserisce un nuovo intero **k** nell'insieme (impostando **a[k]** a **true**). Scrivete la funzione **deleteElement** che elimina dall'insieme l'intero **m** (impostando **a[m]** a **false**). Scrivete la funzione membro **toString()** che restituisce una stringa che rappresenta un insieme come lista di numeri separati da spazi. Visualizzate – per l'insieme vuoto. Scrivete un programma di prova che faccia uso della classe **IntegerSet**. Istanziate diversi oggetti **IntegerSet**. Verificate la correttezza dei metodi.

3) Si scriva una nuova versione della classe **IntegerSet** che ha gli stessi metodi della classe precedenze ma rimuove il limite che gli interi devono essere compresi fra 0 e 99. Utilizzate una ArrayList<Integer> per memorizzare gli elementi dell'insieme e il metodo get o altri metodi che possono tornare utili che trovate già pronti in ArrayList.
   Verificate che il main dell'esercizio 3 funziona anche con la nuova classe **IntegerSet** senza necessità di alcuna modifica (grazie all'incapsulamento).

4) 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*().

5) 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**).

6) 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**).

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

`javac -d . es06src/nomeClasse.java`	*compila e genera il bytecode*
`java nomePackage.nomeClasse`	*esegue il bytecode sulla JVM*