1. Create la classe **PositiveIntegerSet** (insieme di interi positivi). Ogni oggetto di **PositiveIntegerSet** può memorizzare interi positivi.
   Prevedete un costruttore con un varargs di interi e uno che riceve come parametro un **PositiveIntegerSet** e crea un nuovo oggetto uguale al parametro passato.
   Scrivete i metodi che effettuano le comuni operazioni sugli insiemi. In particolare scrivete il metodo ***PositiveIntegerSet union(PositiveIntegerSet s)*** che crea e ritorna un insieme che è l'unione dell'oggetto con l'insieme **s** passato come parametro. Analogamente scrivete i metodi **intersection** e **difference** che creano rispettivamente l'intersezione e la differenza.
   Scrivete il metodo **insertElement** che inserisce un nuovo intero positivo **k** nell'insieme e visualizza un messaggio di errore se si prova a inserire un intero non positivo, il metodo **deleteElement** che elimina dall'insieme un elemento, il metodo **containsElement** che ritorna true o false a seconda che il parametro passato sia presente o meno nell'insieme e il metodo **size** che ritorna il numero di elementi presenti nell'insieme.
   Scrivete il metodo **toString** che restituisce una stringa che rappresenta l'insieme, il metodo **equals** che restituisce *true* se due insiemi contengono gli stessi elementi e *false* altrimenti e un metodo **compareTo** che restituisce 1 se l'insieme ha più elementi dell'insieme passato come parametro, -1 se ne ha di meno e 0 se hanno lo stesso numero di elementi.
   Scrivete infine ogni altro metodo che ritenete utile all'implementazione di quanto richiesto sopra.
   Scrivete un programma di prova che faccia uso della classe **PositiveIntegerSet**. Istanziate diversi oggetti **PositiveIntegerSet**. Verificate la correttezza dei metodi e che venga lanciata un'eccezione quando provate a inserire un elemento non positivo nell'insieme.
   Una volta testata la classe **PositiveIntegerSet** scrivete una nuova classe **SmallIntegerSet** che ha le stesse caratteristiche della **PositiveIntegerSet** ma ammette soltanto valori fra 1 e 1000. Analogamente a quanto fatto prima verificate la correttezza dei metodi e che venga lanciata un'eccezione quando provate a inserire un elemento non valido nell'insieme.

2. Si implementi una gerarchia di classi per rappresentare **Razionale** e **Complex** come classi derivate di una classe astratta **Numero**. Si definisca un'interfaccia **Aritmetica** con le 4 operazioni aritmetiche tra coppie di elementi **Numero** che saranno opportunamente implementate nelle classi **Razionale** e **Complex**. Nel caso in cui si provi a fare operazioni fra elementi di tipo diverso visualizzate un messaggio di errore.
   Verificate il corretto funzionamento con un programma che crea alcuni oggetti delle due classi e fa alcune operazioni aritmetiche.

3. Rivedere l'esercizio 3 dell'esercitazione 7 (**Abbonato**, **AbbonatoPremium**) e lo si implementi utilizzando un array eterogeneo di **Abbonato**.

4. Si riprenda la gerarchia **Persona**, **Professore**, **Studente**, **StudenteTriennale** e **StudenteMagistrale** dell'esercitazione 7. Si scriva una classe per la gestione del corso di laurea. Tale classe utilizzerà un'unica collezione eterogenea per memorizzare i Professori e gli Studenti del corso (al massimo 100). Prevedere un metodo che consenta di stampare il costo totale derivante dal pagamento dei salari dei Professori (cioè la somma dei salari dei Professori) ed un metodo che consenta di stampare il ricavo proveniente dai contributi d'iscrizione versati dagli studenti.

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

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