FONDAMENTI DI PROGRAMMAZIONE B

Tempo a disposizione: 2 ore 15 minuti

Nome ..... Cognome ..... Matricola .....

Esercizio 1 [C++] (15pt). Definire una classe templatica `Stack<T>` che realizza il tipo di dato astratto pila di elementi di tipo `T` (accesso di tipo *LIFO: Last In First Out*). La classe deve definire:

- un costruttore senza argomenti che crea una pila vuota;
- un metodo `push` che aggiunge un elemento di tipo `T` alla pila;
- un metodo `pop` che rimuove dalla pila l'ultimo elemento inserito e lo ritorna come risultato. Il metodo deve lanciare un'eccezione se la pila è vuota.
- un metodo `isEmpty` che controlla se la pila è vuota.
- un metodo `size` che ritorna il numero di elementi nella pila

Non è consentito utilizzare classi della STL. Se necessario, ridefinire gli opportuni metodi, costruttori e/o operatori. Specificare opportunamente eventuali metodi e parametri costanti. Massimizzare incapsulamento e information hiding.

Esercizio 2 [Java] (15pt). Nel contesto degli studenti ed esami universitari, si implementino le seguenti classi ed interfacce.

- Si implementi un'interfaccia `Immatricolabile`, che contiene solamente un metodo senza parametri `getMatricola` che ritorna un oggetto di tipo `String`. Si implementino le classi `StudenteOrdinario` e `StudenteLavoratore`. Entrambe le classi devono implementare l'interfaccia `Immatricolabile`.
- La classe `StudenteOrdinario` rappresenta studenti ordinari non lavoratori. Uno studente ordinario è caratterizzato da una matricola e dal nome della scuola superiore di provenienza. La classe deve sovrascrivere il metodo `equals`. Due studenti ordinari sono uguali per il metodo `equals` se hanno la stessa matricola.
- La classe `StudenteLavoratore` rappresenta studenti lavoratori. Uno studente lavoratore è caratterizzato da una matricola e dal nome dell'azienda in cui lavora. La classe deve sovrascrivere il metodo `equals`. Due studenti lavoratori sono uguali per il metodo `equals` se hanno la stessa matricola e lavorano nella stessa azienda.
- Si implementi la classe `AppelloEsame` che modella un appello di un esame. Un appello d'esame è caratterizzato dal nome dell'esame, dall'anno accademico in cui si svolge l'appello e il numero di iscritti. In ogni appello, è possibile accettare fino a un massimo di 50 iscrizioni per gli studenti ordinari e fino a un massimo di 50 iscrizioni per gli studenti lavoratori. La classe deve definire:
  - un unico costruttore che prende come parametri il nome dell'esame e l'anno accademico in cui si svolge l'appello ed inizializza i campi corrispondenti. Quando costruito, un appello d'esame non ha iscritti;
  - un metodo `iscrivi` che iscrive uno studente all'appello d'esame su cui è invocato il metodo. Se lo studente è già iscritto all'appello d'esame, il metodo deve lanciare un'eccezione di tipo **controllato** `AppelloEsameException`, da implementare. Se l'iscrizione dello studente specificato come parametro viola i vincoli di iscrizione precedentemente descritti, il metodo deve lanciare l'eccezione `AppelloEsameException`;
  - il metodo `equals` (che ridefinisce quello della classe `Object`). Due appelli d'esame sono uguali se hanno lo stesso nome d'esame, lo stesso anno accademico e gli stessi iscritti.
- **(+3pt)** La classe `AppelloEsame` deve implementare l'interfaccia `Comparable<T>`. Il metodo `compareTo` utilizza il numero di studenti lavoratori per il confronto.

N.B. Massimizzare incapsulamento e information hiding. Non è richiesta l'implementazione del metodo `hashCode` per le classi richieste. Per ciascuna classe, è possibile supporre di avere a disposizione una implementazione del metodo `hashCode` coerente col metodo `equals` che implementerete.