FONDAMENTI DI PROGRAMMAZIONE B

Tempo a disposizione: 2 ore 30 minuti

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

Esercizio 1 [C++] (15pt). Definire una classe templatica `Inventory` che rappresenta un inventario di elementi di tipo `T`. La classe deve definire un costruttore senza argomenti che crea un inventario vuoto. Il metodo `add` aggiunge un elemento di tipo `T` all'inventario. Il metodo `count` prende come argomento un elemento di tipo `T` e restituisce il numero di occorrenza dell'elemento passato come argomento. Il metodo `getMostCommon` restituisce l'elemento che occorre più spesso nell'inventario. Un inventario ha dimensione indefinita. Non è consentito usare la STL. Se necessario, ridefinire gli opportuni metodi, costruttori e/o operatori.

```
Inventory<int> inv0;
Inventory<string> inv1;
inv0.add(1); inv0.add(2); inv0.add(1); inv0.add(3);
inv1.add("hello"); inv1.add("hello"); inv1.add("ciao"); inv1.add("hello");

cout << inv0.count(7) << endl; // output: 0
cout << inv0.count(1) << endl; // output: 2
cout << inv1.count("ciao") << endl; // output: 1
cout << inv0.getMostCommon() << endl; // output: 1
```

Esercizio 2 [Java] (15pt). Si consideri un'implementazione di un sistema di gestione degli esami universitari. Ad un esame è possibile iscrivere più studenti ed in seguito verbalizzare il voto di uno studente.

Si implementi la classe `Studente` caratterizzata da nome, cognome e matricola. La classe `Studente` deve implementare un unico costruttore che accetta come parametri il nome, il cognome e la matricola dello studente. I campi della classe `Studente` non devono essere modificali dopo la loro inizializzazione nel costruttore. Inoltre, la classe deve ridefinire i metodi `equals`, `toString` e `hashCode` (non banale). Due studenti sono considerati uguali dal metodo `equals` se hanno la stessa matricola.

Si implementi una classe `Esame` che realizza un esame a cui gli studenti possono iscriversi. La classe `Esame` ha un unico costruttore senza parametri che crea un esame senza studenti iscritti e senza studenti che hanno verbalizzato l'esame.

- Il metodo `iscrivi` ha tipo di ritorno `void` e prende come parametro uno studente e lo iscrive all'esame. Il metodo deve lanciare un'eccezione **controllata** `StudenteGiaIscrittoException`, da implementare, se lo studente è già iscritto all'esame.
- Il metodo `verbalizza` accetta come parametri uno studente e un voto da 0 a 30. Il metodo deve lanciare un'eccezione **controllata** `StudenteNonIscrittoException`, da implementare, se lo studente non è iscritto all'esame. Inoltre, deve lanciare un'eccezione **controllata** `StudenteGiaVerbalizzatoException`, da implementare, se lo studente ha già verbalizzato l'esame.

Si utilizzi la seguente classe già implementata `Verbalizzazione`

```
public class Verbalizzazione {
  private final Studente studente;
  private final int esito;

  public Verbalizzazione(Studente studente, Esito esito) {
    this.studente = studente;
    if (esito < 0 || esito > 30) throw new RuntimeException();
    this.esito = esito;
  }
  public Studente getStudente() {  return studente; }

  public Esito getEsito() { return esito; }

  // una Verbalizzazione e' equals a un'altra che abbia stesso esito e stesso studente
  @Override
  public boolean equals(Object other) { ... }
```

```
  // implementazione del metodo hashCode coerente con equals
  @Override public int hashCode() { ... }

  @Override public String toString() {
    return studente.toString() + "␣Esito:␣" + esito;
  }
}
```